Dérivation systématique d'un algorithme de segmentation d'images . Un exemple d'application du formalisme GAMMA

Christian Creveuil, Gersan Moguerou

INRIA

UNITÉ DE RECHERCHE
INRIA-RENNES

Institut National
de Recherche
en Informatique
et en Automatique

Domaine de Voluceau
Rocquencourt
B.P.105
78153 Le Chesnay Cedex
France
Tél.:(1) 39 63 55 11

Rapports de Recherche

N°1049

Programme 6

DERIVATION SYSTEMATIQUE D'UN ALGORITHME DE SEGMENTATION D'IMAGES UN EXEMPLE D'APPLICATION DU FORMALISME GAMMA

Christian CREVEUIL
Gersan MOGUEROU

Juin 1989

RR-1049

IRISA

INSTITUT DE RECHERCHE EN INFORMATIQUE
ET SYSTÈMES ALÉATOIRES

Campus Universitaire de Beaulieu
35042 - RENNES CÉDEX
FRANCE
Téléphone : 99 36 20 00
Télex : UNIRISA 950 473 F
Télécopie : 99 38 38 32

DERIVATION SYSTEMATIQUE D'UN ALGORITHME DE SEGMENTATION D'IMAGES

Un exemple d'application du formalisme GAMMA

Publication Interne n° 472 - Mai 1989 - 46 Pages

Christian CREVEUIL (*) Gersan MOGUEROU (**)

(*) IRISA
Campus de Beaulieu 35042 RENNES CEDEX
FRANCE

(**) IRISA
INSA 20 avenue des Buttes de Coësmes 35043 RENNES CEDEX
FRANCE

Résumé: Nous montrons dans cet article qu'il est possible d'utiliser une méthode formelle de dérivation de programmes pour résoudre des problèmes plus compliqués que les "problèmes d'école" généralement traités. L'application qui nous intéresse est un problème de traitement d'images dont le but est la reconnaissance de la topographie tridimensionnelle du réseau vasculaire cérébral à l'aide de deux radiographies, prises sous deux angles différents. Nous dérivons ici la partie segmentation de cette application; la segmentation consiste à découper en segments les graphes de points caractéristiques extraits des deux radiographies, le but étant de réduire la combinatoire du problème. La dérivation s'effectue à partir des spécifications du problème exprimées en logique du 1er ordre, et est basée sur la conservation de propriétés invariantes. Le programme est dérivé dans un formalisme de haut niveau, appelé GAMMA; la structure de données principale de ce formalisme est le multi-ensemble, et l'exécution d'un programme GAMMA peut être vue comme une succession de réactions chimiques consommant des éléments du multi-ensemble, qui vérifient une certaine condition, pour en produire de nouveaux. Le non-déterminisme de ce modèle de calcul et l'aspect local des opérations effectuées sur les éléments du multi-ensemble autorise de plus une mise en œuvre parallèle.

.../...

SYSTEMATIC DERIVATION OF AN IMAGE SEGMENTATION ALGORITHM

An example of application of the GAMMA formalism

Abstract: In this paper, we show that a formal method for program derivation can be used to solve more difficult problems than the generally discussed "toy problems". We study an image processing application that deals with the recognition of the tridimensionnal topography of the vascular cerebral network; this recognition is carried out from two radiographies, taken under two different angles. We present here the derivation of the segmentation part of this application; the segmentation consists of cutting up the graphs of characteristic points extracted from the two radiographies, in order to reduce the combinatory. The derivation is carried out from the problem specification expressed in first order logic, and is based on the preservation of invariance properties. The program is derived in a high level formalism, called GAMMA; the main data structure is the multiset, and the execution of a GAMMA program can be seen as a succession of chemical reactions consuming elements from the multiset, and producing new elements according to specific rules. Furthermore, the non-determinism of the model, and the locality of the operations involving the elements of the multiset allow a parallel implementation.

1. Introduction.

Plusieurs méthodes formelles de dérivation de programmes ont vu le jour ces dernières années [Dijkstra 76] [Gries 81]. Une critique habituelle de ces méthodes est qu'elles ne sont utilisables que sur des "problèmes d'école"; la complexité engendrée par la dérivation d'un programme ne permet pas de traiter des applications réelles. Cette difficulté est due aux langages de programmation dans lesquels sont dérivés les programmes. Plus précisément, les structures de données et de contrôle de ces langages sont souvent trop contraignantes: elles imposent une séquentialisation superflue sans rapport avec les propriétés logiques du problème à résoudre. C'est le cas avec les langages impératifs qui utilisent un flot de contrôle séquentiel et explicite dans les programmes. De même, les langages fonctionnels utilisent de façon intensive la récursivité dans les structures de données (listes, arbres) et dans les programmes. Cela entraîne généralement une spécification excessive du contrôle. Prenons l'exemple de la recherche du plus grand élément d'un ensemble; l'ensemble sera représenté par une liste et le programme s'écrira de la façon suivante:

f(l)=**si** cdr(l)=nil **alors** car(l) **sinon** max(car(l),f(cdr(l))) **fsi**.

Ce parcours séquentiel de la liste n'est pas imposé par la spécification du problème (les éléments peuvent a priori être comparés dans n'importe quel ordre, et même en parallèle), mais par la nature du langage (et notamment de ses structures de données).

Afin de remédier à ces inconvénients, un formalisme de haut niveau, appelé GAMMA, a été développé à l'IRISA [Banâtre 86]. Pour éviter une séquentialisation excessive dans les programmes, il faut proscrire l'utilisation de tableaux ou de structures de données récursives, trop contraignantes du point de vue du contrôle; GAMMA utilise donc le type de donnée le moins structuré possible, le multi-ensemble, qui est identique à un ensemble, mis à part qu'il peut contenir plusieurs occurences d'un même élément. D'autre part, la structure de contrôle associée doit être adaptée à cette structure de données; un multi-ensemble n'étant muni d'aucune structure, les programmes doivent pouvoir être exécutés de façon "chaotique". L'idée est de prendre comme modèle d'exécution la réaction chimique: un multi-ensemble peut être vu comme un ensemble de molécules; l'exécution d'un programme se ramène alors à une

succession de réactions chimiques: tant qu'une "condition de réaction" entre des molécules est vraie, ces dernières sont remplacées par les molécules résultant de la réaction. Les réactions se font de façon non-déterministe et plusieurs réactions sur des éléments disjoints peuvent même s'effectuer en parallèle. Dans le formalisme GAMMA, la condition de réaction est représentée par une fonction booléenne R, et les données résultant de la réaction sont produites par une fonction A, appelée action. La succession des réactions chimiques est assurée par l'opérateur Γ qui prend en argument R et A. L'application de $\Gamma(R,A)$ à un multi-ensemble consiste alors en une succession d'opérations effectuées de façon non déterministe sur des n-uplets $(x_1, \ldots, x_n)$ d'éléments du multi-ensemble; chaque opération entraîne le remplacement de $\{x_1, \ldots, x_n\}$ tel que $R(x_1, \ldots, x_n)$ par les éléments produits par $A(x_1, \ldots, x_n)$. Le calcul s'arrête quand plus aucune opération ne peut être effectuée. Le non-déterminisme de GAMMA et l'aspect local des opérations sur les n-uplets d'éléments a permis de développer une mise en œuvre parallèle [Banâtre 87, 88].

La dérivation d'un programme GAMMA s'effectue à partir de la spécification du problème exprimée en logique du 1er ordre. Cette spécification est scindée en deux conjonctions de formules: un invariant (I) et un variant (V); (I) est constitué des propriétés vérifiées par l'état initial. La condition de réaction s'obtient directement de la négation du variant et l'action que l'on dérive doit assurer la stricte monotonie d'une fonction de terminaison et la conservation de l'invariant. Nous montrons dans cet article qu'il est possible de dériver de façon systématique à partir de ses spécifications, un algorithme complexe de segmentation d'images. Cet algorithme s'inscrit dans le cadre d'un projet de stéréovision dont le but est la reconnaissance de la topographie tridimensionnelle du réseau vasculaire cérébral à travers un couple de radiographies [Camillerapp 87]. Les vues radiographiques sont prises avec un angle de parallaxe de 6° et sont numérisées; chaque numérisation produit une matrice de pixels. Les deux images sont analysées indépendamment l'une de l'autre et un graphe de points caractéristiques représentant les vaisseaux sanguins est extrait de chaque image. Le problème de la stéréovision consiste ensuite à mettre en correspondance les points de chaque graphe, c'est-à-dire trouver pour chaque point de chaque graphe son homologue dans l'autre

graphe, afin de déterminer la troisième coordonnée spatiale. Pour des raisons évidentes de complexité d'algorithme, on ne peut essayer tous les appariements. Il convient donc de réduire l'espace de recherche des points homologues. Cependant cette réduction de la complexité n'est pas suffisante et, afin de lever de nombreuses ambiguïtés lors de la mise en correspondance, chaque vaisseau (liste de ses points caractéristiques dans le graphe) est découpé en segments (sous-liste de ces points). C'est ce processus de segmentation que nous dérivons ici.

Nous présentons le formalisme GAMMA dans le chapitre 2 et la méthode de dérivation associée dans le chapitre 3. La dérivation de l'algorithme de segmentation est décrite dans le chapitre 4. Enfin, dans le chapitre 5, nous exposons un schéma d'évaluation parallèle de GAMMA ainsi que des procédés permettant de rendre plus efficace l'exécution des programmes.

2. Le formalisme GAMMA.

Nous donnons d'abord une définition formelle de l'opérateur Γ tel que nous l'avons présenté plus haut. Nous montrons ensuite que cette définition peut se généraliser à plusieurs couples (condition de réaction, action).

2.1. L'opérateur Γ.

L'élément principal du formalisme GAMMA est l'opérateur Γ qui prend en paramètre un doublet (R,A). R, la condition de réaction, est une fonction booléenne qui prend en argument un n-uplet d'éléments du multi-ensemble traité; lorsque la condition de réaction est vraie, on applique à ce n-uplet l'action A, qui produit le multi-ensemble résultat de la réaction. La sémantique du programme GAMMA qui en résulte est la suivante:

$$\Gamma(R, A)\ (M) \equiv \textbf{si}\ \exists\ x_1, \ldots, x_n \in M \text{ tel que } R(x_1, \ldots, x_n)$$
$$\textbf{alors}\ \Gamma(R, A)\ ((M - \{x_1, \ldots, x_n\}) \cup A(x_1, \ldots, x_n))$$
$$\textbf{sinon}\ M$$
$$\textbf{fsi}$$

Les composants atomiques du multi-ensemble peuvent être des entiers, des réels, des caractères, des n-uplets, des multi-ensembles, etc.

Notons quelques propriétés importantes de GAMMA:

- le non-déterminisme d'exécution: si plusieurs n-uplets vérifient la condition de réaction, le choix effectué entre eux est non-déterministe.

- le non-déterminisme de résultat: différentes exécutions d'un même programme sur des données identiques peuvent aboutir à des résultats différents. Cependant, on s'intéresse principalement à des programmes déterministes (de résultat).

- la localité: le calcul de la condition de réaction et l'application de l'action sur un n-uplet d'éléments s'effectuent indépendamment du reste des éléments.

- le parallélisme, qui découle de la localité: plusieurs réactions faisant intervenir des n-uplets disjoints peuvent s'effectuer simultanément.

Donnons un exemple de calcul des nombres premiers:

nombres_premiers (n) = Γ(R,A) ({2, ... , n}) avec

$\mathbf{R(x_1,x_2)}$ **= multiple** $\mathbf{(x_1,x_2)}$

$\mathbf{A(x_1,x_2)} = \{x_2\}$

où multiple (x,y) est vrai si et seulement si x est multiple de y.

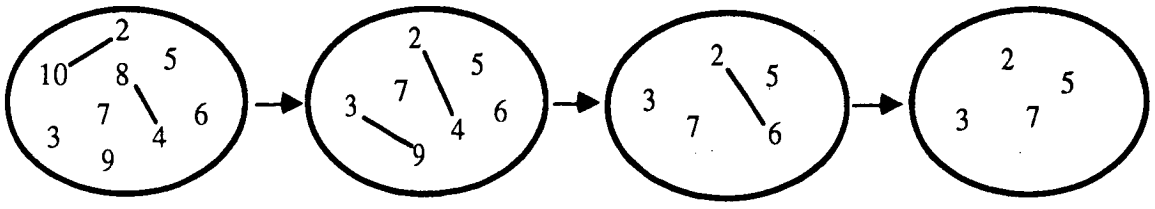

fig.1 Exécution possible de nombres_premiers(10). Un trait reliant deux éléments x et y exprime que ceux-ci réagissent.

2.2. Généralisation.

La définition de l'opérateur Γ se généralise à m doublets (R_i,A_i) de la façon suivante:

$\Gamma((R_1, A_1), ..., (R_m, A_m))\ (M) \equiv$

cas

$\exists\ x_1, \ldots, x_{n1} \in M$ tel que $R_1 (x_1, \ldots, x_{n1}) \rightarrow$

$\Gamma((R_1, A_1), ..., (R_m, A_m))\ ((M - \{x_1, \ldots, x_{n1}\}) \cup A_1 (x_1, \ldots, x_{n1})),$

...

$\exists\ x_1, \ldots, x_{nm} \in M$ tel que $R_m (x_1, \ldots, x_{nm}) \rightarrow$

$\Gamma ((R_1, A_1), ..., (R_m, A_m))\ ((M - \{x_1, \ldots, x_{nm}\}) \cup A_m (x_1, \ldots, x_{nm})),$

autre cas M

fcas

Si plusieurs conditions de réaction sont vérifiées en même temps, le choix effectué entre elles est non-déterministe.

Donnons en exemple le calcul du pgcd:

pgcd(M) = $\Gamma((R_1, A_1), (R_2, A_2))$ (M) avec

$\mathbf{R_1\ (x_1,x_2) = (x_1 \geq x_2) \wedge (x_2 \neq 0)}$

$\mathbf{A_1\ (x_1,x_2) = \{x_1\ mod\ x_2, x_2\}}$

$\mathbf{R_2\ (x) = (x=0)}$

$\mathbf{A_2\ (x) = \{\}}$

La première réaction exprime le fait que $pgcd(x_1,x_2)=pgcd(x_1 \bmod x_2,x_2)$ si $x_1 \geq x_2$ et $x_2 \neq 0$. La deuxième réaction élimine les "0" introduits par A_1 quand x_1 est multiple de x_2.

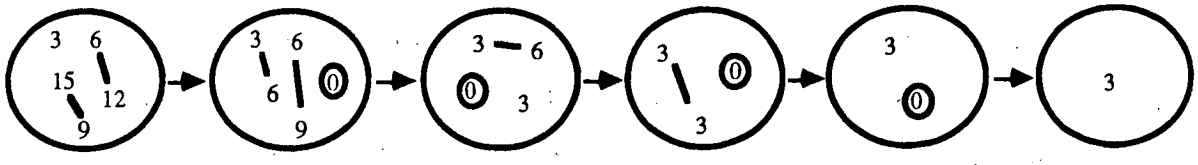

fig. 2 Exécution possible de pgcd:{3,15,6,12,9}. Un trait reliant deux éléments x_1 et x_2 dénote que ceux-ci vérifient $R_1(x_1,x_2)$; un cercle entourant un élément x dénote que celui-ci vérifie $R_2(x)$.

3. Dérivation de programmes GAMMA.

Afin d'introduire la méthode de dérivation [Coutant 89], nous présentons d'abord informellement une dérivation du programme de calcul des nombres premiers présenté plus haut.

3.1. Calcul des nombres premiers.

Le programme à dériver doit prendre en argument un multi-ensemble {2, ... , n} et rendre en résultat un multi-ensemble M qui contient tous les nombres premiers compris entre 2 et n. Soit (S) la spécification du résultat dans le langage de la logique du premier ordre:

(1) $M : \{ent\}$

(2) $\forall x \in M , x \in \{2, ... , n\}$

(3) $\forall x \in \{2, ... , n\}\ [(\forall y \in \{2, ... , n\}\ \neg \text{multiple}(x,y)) \Rightarrow x \in M]$

(4) $\forall x, \forall y \in M\ \neg \text{multiple}(x,y)$

Ces formules signifient que: (1) M est un multi-ensemble d'entiers, (2) M ne contient que des éléments de {2, ... , n}, (3) tous les nombres premiers de {2, ... , n} appartiennent à M, (4) chaque élément de M est un nombre premier.

On cherche d'abord à décomposer $(S)\equiv(1)\wedge(2)\wedge(3)\wedge(4)$ en deux formules (I) et (V) telles que $(S)\equiv(I)\wedge(V)$. (I), l'invariant, est une conjonction de propriétés qui reste vraie tout au long du calcul. (V), le variant, est constitué des propriétés qui ne sont vraies qu'à la fin du calcul. Ici, nous choisissons $(I)\equiv(1)\wedge(2)\wedge(3)$ car on peut facilement valider $(1)\wedge(2)\wedge(3)$ en choisissant M={2, ... , n} à l'initialisation. Le calcul de M se poursuit tant que $(V)\equiv(4)$ n'est pas vrai, c'est-à-dire tant que la formule $\neg(V)\equiv\exists x\ \exists y \in M$ multiple(x,y) est vraie. Cela s'exprime par la condition de réaction R(x,y)=multiple(x,y). Le but étant de rendre (V) vrai, il faut maintenant trouver une action A associée à R qui préserve (I) et qui fasse strictement décroître une fonction de terminaison f bornée inférieurement. Prenons f(M)=nombre de n-uplets pouvant réagir; f est bornée inférieurement par 0. Pour conserver (I), la seule action possible est A(x,y)={y}; en effet, on ne peut ni ajouter de valeurs (2), ni retirer de M la valeur y qui peut être un nombre premier. De plus, en éliminant x, on élimine un multiple de M, et f(M) décroît

d'au moins 1. D'où le programme:

nombres_premiers (n) = Γ(R,A) ({2, ... , n}) avec

R(x,y) = multiple (x,y)

A(x,y) = {y}

3.2. La méthode de dérivation.

La méthode présentée se compose de quatre grandes étapes, en plus de la spécification du problème.

Etape 0: Spécification du problème.

Le problème à résoudre est défini par une spécification, exprimée en logique du premier ordre. Cette spécification décrit le type des données ainsi que les propriétés des données et du résultat.

Etape 1: Scission de la spécification.

De la spécification (S) on isole un invariant (I) et un variant (V) tels que $(S)\equiv(I)\wedge(V)$. (I) est généralement choisi comme la partie de (S) qui est la plus facile à établir par une initialisation. Par conséquent, les choix de (I) et de l'état initial sont fortement liés. Notons que cette initialisation peut être réalisée également par un programme GAMMA et qu'on obtient ainsi une suite de compositions de programmes GAMMA.

Etape 2: Condition de réaction.

A partir du variant on établit la (ou les) condition(s) de réaction. Pour ce faire, on met (V) sous forme prénexe; quand $\neg(V)$ ne contient aucun quantificateur universel ($\forall$) on dérive immédiatement la (ou les) condition(s) de réaction. Par exemple, de $\neg(V)\equiv\exists x\ \exists y \in M\ \neg C(x,y)$ on dérive $R(x,y)=\neg C(x,y)$. Si $C(x,y)=C_1(x,y)\wedge C_2(x,y)$ alors $\neg C(x,y)=\neg C_1(x,y)\vee\neg C_2(x,y)$; on dérive alors deux conditions de réactions: $R_1(x,y)=\neg C_1(x,y)$ et $R_2(x,y)=\neg C_2(x,y)$.

Etape 3: Fonction de terminaison.

Afin d'assurer la terminaison du programme, on associe au variant une fonction de terminaison f qui doit être strictement monotone et bornée sur un ordre bien fondé. En pratique, f est souvent définie comme étant le nombre $N_R(M)$ de n-uplets $(x_1, \ldots, x_n)$

vérifiant $R(x_1,..., x_n)$.

Etape 4: Action.

Enfin, pour chaque condition de réaction, il faut trouver une action qui assure la stricte monotonie de f et la conservation de l'invariant (I).

Remarquons qu'une configuration (I,V) ne conduit pas nécessairement à un programme; la dérivation peut échouer par exemple à l'étape 2 parce qu'un $\forall$ est apparu dans la négation du variant, ou à l'étape 4 quand aucune action ne peut être trouvée. Il convient alors de revenir sur des choix antérieurs:

- essayer une autre fonction de terminaison (étape 3).
- reformuler le couple (I,V) en un couple (I,V') tel que $(I)\wedge(V) \Leftrightarrow (I)\wedge(V')$ (étape 2).
- considérer une autre décomposition (I',V') (étape 1).
- choisir un nouvel état initial (étape 1).

Cette méthode a été utilisée pour dériver systématiquement un certain nombre de programmes GAMMA (programmes de tris, de parcours de graphes, ...) [Coutant 89]. Nous présentons dans le prochain chapitre son application à la dérivation d'un algorithme de segmentation d'images.

4. Dérivation d'une application image.

L'algorithme présenté ici s'inscrit dans le cadre d'un projet visant à automatiser la reconnaissance en trois dimensions du réseau vasculaire cérébral à partir de deux vues radiographiques [Camillerapp 87]. Après avoir présenté le problème de façon informelle, nous le spécifions puis nous dérivons un algorithme dans le formalisme GAMMA [Moguérou 88].

4.1. Problème général de la stéréovision.

Nous cherchons à modéliser en trois dimensions le réseau vasculaire cérébral à partir de deux radiographies du cerveau. Un exemple de vue radiographique est donné en figure 3.

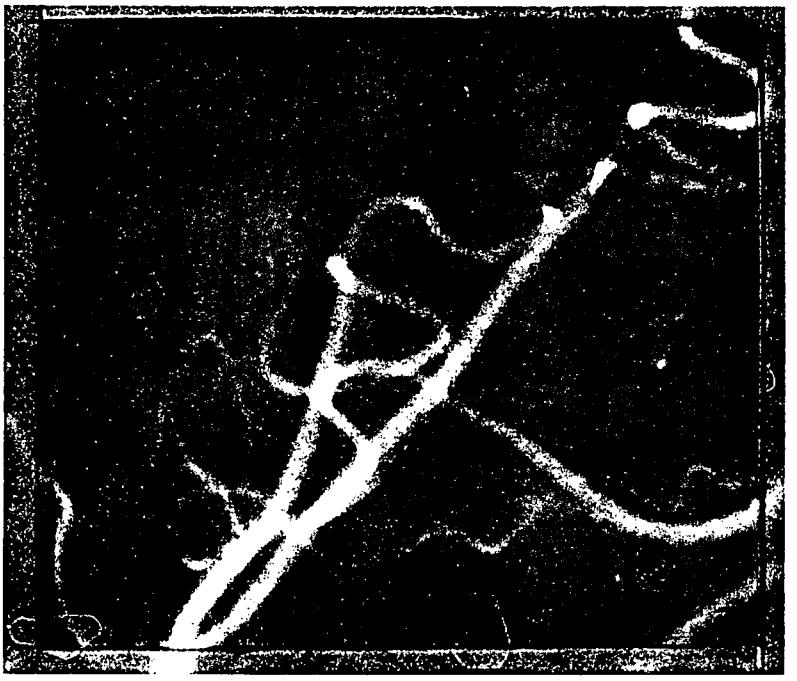

fig.3 exemple d'une vue radiographique en niveaux de gris.

Connaissant les projections M' et M" d'un point M de l'espace dans les deux plans image, nous savons calculer les coordonnées spatiales de M à partir des coordonnées planaires de M' et M" et de la parallaxe, définie comme étant l'angle entre les deux prises de vue. Le problème consiste donc à trouver, pour chaque point M' d'un plan image, son correspondant M" dans l'autre plan afin d'en déduire M. Mais la taille des images traitées ne permet pas d'essayer toutes les mises en correspondances. Par exemple, dans le cas d'une image 350x350 pixels, il faudrait évaluer, pour chacun des 122500 points d'une image, les 122500 correspondances possibles avec les points de l'autre image.

Toutefois, il est possible de diminuer notablement cette combinatoire. Premièrement, des informations a priori sur la géométrie de la prise de vue permettent de réduire l'espace de recherche d'un plan à un segment de droite: d'une part, l'ensemble des points M" correspondants possibles d'un point M' est une droite, appelée droite épipolaire [Faugeras 88], et comme les radiographies sont prises en déplaçant la source et en laissant le plan image invariant, les droites épipolaires sont les lignes de balayage; d'autre part, la connaissance de la parallaxe et de la profondeur de la scène observée réduit la zone de recherche à un segment de

droite sur la ligne épipolaire. Grâce à ces deux considérations géométriques, le nombre de correspondants possibles pour chacun des 122500 points d'une image se limite à 50 points dans l'autre image. Deuxièmement, en extrayant de chaque image un graphe de points caractéristiques, on diminue le nombre de points pour lesquels on cherche un correspondant. Cet algorithme, appelé squelettisation, n'est pas détaillé ici, le lecteur intéressé en trouvera une description complète dans [Leplumey 86]. Ainsi, d'une matrice de 122500 points on extrait un graphe d'environ 2000 points et pour chacun des points, il n'y a plus, en moyenne, que 3 ou 4 correspondances à essayer dans la fenêtre de recherche. La figure 4 montre un exemple de graphes extraits de deux radiographies.

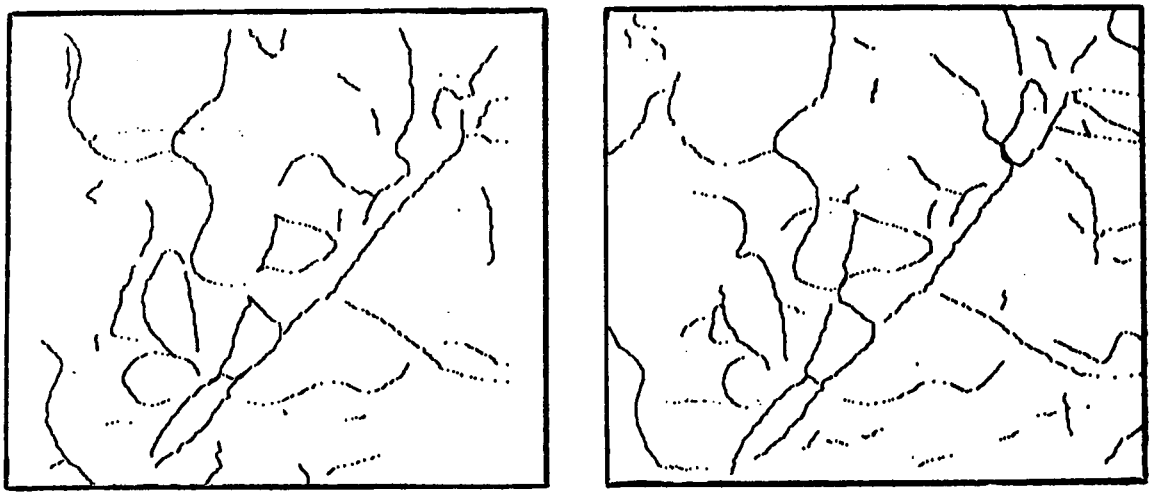

fig.4 les graphes A et B extraits de deux radiographies.

A l'issue de la squelettisation, la notion d'image est réduite au graphe extrait et on a les propriétés suivantes:

1. Un graphe est un ensemble de vaisseaux; chaque vaisseau est une suite de points liés entre eux par les relations successeur et prédécesseur.

2. Deux points consécutifs d'un même vaisseau se trouvent sur deux lignes différentes (pas de plat) et sont sur des lignes adjacentes (pas de hachage).

3. Les images sont recentrées (figure 5): soient un point M(x,y,z) de l'espace et M'(x',y') sa projection dans une image; l'homologue M"(x",y") de M', s'il existe, se trouve

dans l'autre image dans une fenêtre de longueur TF=50 pixels centrée en M'. On a: y'=y" et |x'-x"| ≤ TF/2.

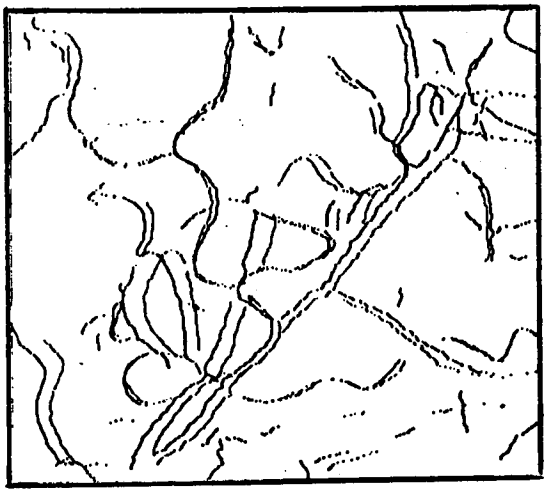

fig.5 les deux graphes superposés.

4.2. Corrélation.

A un point de l'espace peut correspondre 0, 1 ou 2 points projetés. Le cas où il y en a deux (un dans chaque image) est évidemment le plus souhaitable. Mais il n'en est pas toujours ainsi. Ces lacunes peuvent être dues à:

- une numérisation imparfaite,

- une squelettisation imparfaite due à la présence de bruit ou à un contraste insuffisant sur la radiographie,

- des informations non rendues par la squelettisation: le chaînage en liste par les liens successeur et prédécesseur empêche la détection des recouvrements et des embranchements.

Pour effectuer la mise en correspondance, nous utilisons les règles de Marr [Marr 82]:

1. contrainte d'unicité: Chaque point d'une image a au plus un correspondant dans l'autre image. Il ne peut pas en avoir plus d'un car la projection d'un point de l'espace dans un plan est unique.

2. règle de continuité: Les objets de la scène observée ont en général des contours ou des surfaces continues. La disparité entre deux points homologues étant ici la valeur

absolue de la différence de leurs abscisses, la règle de continuité impose que les disparités entre deux couples de points projection varient également continûment. Etant donnés deux points M' et M" que l'on a mis en correspondance, cela revient à privilégier l'appariement entre des couples de points P' et P" voisins de M' et M" si la disparité entre P' et P" est voisine de celle entre M' et M".

Mais la combinatoire de la corrélation reste élevée à cause des ambiguïtés sur le choix des meilleures correspondances. Nous montrons maintenant qu'il est possible de regrouper les points en segments pour réduire encore cette combinatoire.

4.2.1. Segmentation.

On sait que l'homologue d'un point M' = (x',y') d'une image, s'il existe, se trouve dans une fenêtre de largeur 50 pixels centrée en (x',y') dans l'autre image. Mais il peut y avoir plusieurs points dans cette fenêtre. Il faut réaliser l'appariement entre M' et un point M" qui ne contredit pas les autres appariements: les décisions locales doivent être compatibles avec des propriétés globales de cohérence. Nous regroupons donc les points en régions de décision homogène. En effet, les régions contiennent des informations contextuelles plus riches que les points isolés, ce qui permet de limiter encore les candidats à la corrélation ainsi que les ambiguïtés [Long 85]. Les graphes étant structurés en vaisseaux, ici, ces régions seront des parties de vaisseaux. Nous décrivons dans ce paragraphe le découpage des vaisseaux en "segments". La mise en correspondance de ces segments à l'aide de règles est introduite dans le paragraphe suivant.

Soit P un point d'un vaisseau V, on note PC(P) l'ensemble des points candidats à la mise en correspondance avec P:

PC (P(x,y)) = { P'(x',y') de l'autre image | y = y' et | x-x'| ≤ TF/2 }.

Soient deux points voisins P_1 et P_2 d'un même vaisseau, on note $EV(P_1,P_2)$ l'environnement de vaisseaux de P_1 par rapport à P_2:

$EV(P_1,P_2) = \{ P'_1 \in PC(P_1) \mid P'_1$ n'a pas de successeur ou le successeur de P'_1 n'est pas candidat de $P_2 \} \cup$

$\{ P'_2 \in PC(P_2) \mid P'_2$ est le successeur d'un candidat de $P_1 \}$.

$EV(P_1,P_2) = EV(P_2,P_1)$ signifie que chaque vaisseau V' de l'autre image, qui coupe les fenêtres de P_1 et de P_2, coupe celles-ci en exactement deux points voisins P'_1 et P'_2. La figure 6 montre des exemples de vaisseaux V', V" qui coupent les fenêtres de P_1 et P_2 en plus (ou moins) de deux points: $EV(P_1,P_2) = \{P"_1\} \cup \{P'_2,P'_4\}$ et $EV(P_2,P_1) = \{P'_4\} \cup \{P'_3\}$.

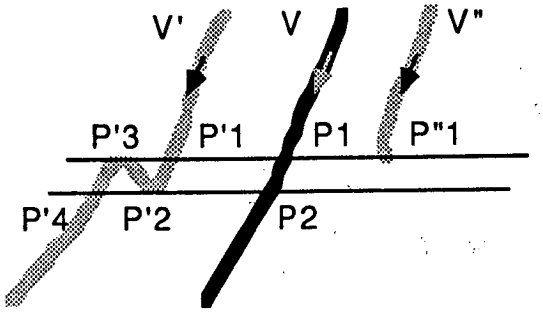

fig.6 Exemples de vaisseaux ne coupant pas les fenêtres de P_1 et P_2 en 2 points exactement. Conventions: les vaisseaux appartenant à des images différentes sont en différents gris; le sens (arbitraire) de parcours de la relation successeur est représenté par une flèche; les fenêtres de P1 et P2 sont les traits horizontaux.

La segmentation de l'image B par l'image A est réalisée de la façon suivante:

Soit V un vaisseau de A et P_1 un point de V. Deux cas peuvent se produire,

- soit P_1 est extrémité de V; dans ce cas P_1 est également extrémité de segment,
- soit P1 a un successeur P_2; si $EV(P_1,P_2) \neq EV(P_2,P_1)$, on segmente entre P_1 et P_2.

Dans le cas de la figure 7, $EV(P_1,P_2)$ $(= \{P'_1\}) \neq EV(P_2,P_1)$ $(= \varnothing)$ car V' ne traverse pas la fenêtre de P_2.

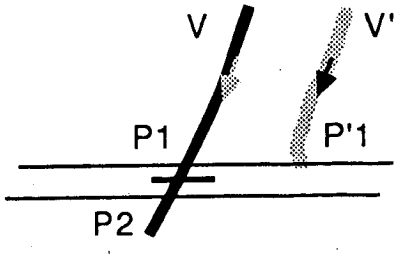

fig.7 segmentation entre P_1 et P_2

On réalise de même la segmentation de l'image B par l'image A.

Enfin on fait l'intersection des deux segmentations, c'est à dire que si l'on a segmenté entre P'_1 et P'_2, alors on segmente entre tous les couples (P_1,P_2) tels que P_1 et P_2 sont voisins et $P_1 \in PC(P'_1)$ et $P_2 \in PC(P'_2)$. La figure 8 montre un exemple d'intersection des segmentations. Supposons que P_2 soit le successeur de P_1; si P'_2 est le successeur de P'_1, on a $EV(P_1,P_2) = EV(P_2,P_1) = \{P'_2\}$, on n'a donc pas segmenté. Par contre, comme $EV(P'_1,P'_2)$ $(= \{P_2\}) \neq EV(P'_2,P'_1)$ $(= \{P_2,P_3\})$, on segmente entre P'_1 et P'_2. En faisant l'intersection, on propage la segmentation au couple (P_1,P_2).

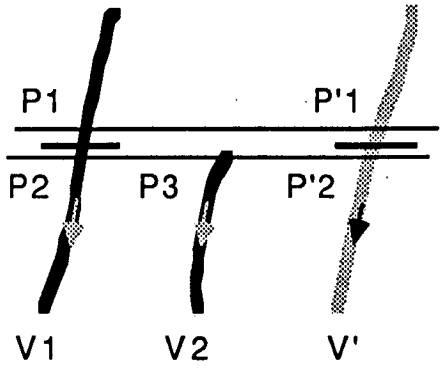

fig.8 intersection des segmentations

Cette intersection peut propager la segmentation le long de toute l'image tant que l'on peut passer ainsi alternativement d'un graphe à l'autre grâce à la relation PC. Intéressons-nous à la fermeture réflexive et transitive PC* de cette relation. Etant donné un point P_1 quelconque, $PC^*(P_1)$ est, par définition, l'ensemble des points que l'on peut joindre par transitivité grâce à la relation PC. $PC^*(P_1)$ comprend donc tous les points qui doivent être considérés pour décider de la segmentation entre ce point P_1 et ses voisins dans son vaisseau. Plus précisément, si P_2 est le successeur ou le prédécesseur de P_1, les ensembles $PC^*(P_1)$ et $PC^*(P_2)$ forment le contexte de décision de l'algorithme de segmentation.

La connaissance de ce contexte permet de résoudre la segmentation à l'aide de deux algorithmes complémentaires. Le premier procède par découpage des vaisseaux en segments comme on vient de l'indiquer. Le second consiste à regrouper les points voisins tels que l'on a

égalité entre les environnements de vaisseaux pour ces couples aussi bien que pour tous les couples de leur contexte PC*.

4.2.2. Mise en correspondance des segments.

Une fois les segments déterminés, il convient de les mettre en correspondance. De la même façon que l'on a défini les ensembles de points candidats PC(P), nous définissons l'ensemble des segments candidats du segment S; S' est candidat de S si chaque point de S a un candidat parmi les points de S'. On a les propriétés suivantes entre deux segments S et S' candidats:

1. S et S' ont même nombre de points. La mise en correspondance des points candidats respectifs est donc immédiate.

2. Les extrémités de S et S' sont en correspondance ligne à ligne.

Ces propriétés permettent de définir la vraisemblance de l'appariement de deux segments S et S':

$$\text{vrais}\ (S,S') = N * \sum_{i=1}^{N} (x_i - x'_i)^2 - \left(\sum_{i=1}^{N} (x_i - x'_i) \right)^2$$

où: - N = longueur du segment (en nombre de points)

- x_i = abscisse du ième point de S

- x'_i = abscisse du ième point de S'

Ce critère, analogue à une variance, a d'autant plus de signification que N est grand et n'en a aucune quand N=1. Il permet de classer les segments S' candidats de S et la mise en correspondance est ainsi facilitée. La règle de continuité de Marr aide à lever des ambiguïtés sur des segments réduits à un point. D'autres règles permettent de traiter les cas de recouvrements, de créer des liens virtuels entre vaisseaux et de contrôler la continuité des disparités [Camillerapp 87]. Après la mise en correspondance des segments, la dernière étape consiste à calculer la troisième dimension.

4.3. Dérivation du programme GAMMA.

Nous nous proposons de dériver à partir de ses spécifications un programme GAMMA de segmentation d'images.

4.3.1. Spécifications de la segmentation.

Les spécifications constituent une formalisation des propriétés décrites dans le paragraphe 4.2.1. Elles sont adaptées à la structure de données utilisée par le formalisme: le multi-ensemble. Chaque point est représenté par un triplet (image, coordonnées, successeur) où successeur permet de chaîner entre eux les points d'un même vaisseau. Nous définissons dans une syntaxe à la Pascal les types suivants:

image = A | B

coord = (x,y : ent)

$point_0$ = (i : image, co, succ : coord)

Soit M_0 le multi-ensemble initial, il est constitué des points composant les graphes résultats de la squelettisation. On a les spécifications suivantes:

M_0 : { $point_0$ } (1)

Les points d'un même graphe ont des coordonnées différentes:

$\forall P \in M_0 \ \forall P' \in M_0 - \{ P \} \quad P.i = P'.i \Rightarrow P.co \neq P'.co$ (2)

Un graphe est structuré en vaisseaux qui sont des suites de points liés entre eux par la relation successeur:

$\forall P_1 \in M_0$ (3)

$P_1.co \neq NIL \ \wedge$

$[P_1.succ \neq NIL \Rightarrow$

$[\exists! P_2 \in M_0 - \{ P_1 \}$

$P_2.i = P_1.i \ \wedge$

$P_1.succ = P_2.co \ \wedge$

$P_2.co.y = P_1.co.y \pm 1$

$] \quad]$

où NIL désigne une valeur constante de type coord signifiant la fin de la liste.

Soient P et P' deux points de M_0; afin d'alléger l'écriture des spécifications, nous utilisons l'abréviation P'∈ PC(P), qui indique que P' est un candidat de P, à la place de:

$$P.i \neq P'.i \wedge P.co.y = P'.co.y \wedge | P.co.x - P'.co.x | \leq TF/2.$$

Soit M le multi-ensemble résultat de la segmentation de M_0.

Les points de M ont deux champs supplémentaires P.pc* et P.seg: P.pc* représente l'ensemble PC*(P) défini au paragraphe précédent; P.seg est le successeur de P si P et son successeur sont dans le même segment et NIL sinon. Leur type est défini par:

$$point = (i : image, co, succ : coord, pc^* : \{point_0\}, seg : coord)$$

d'où la structure de M:

$$M : \{ point \} \qquad (4)$$

La correspondance entre M_0 et M est donnée par:

$$\{ (i,co,succ) \in M_0 \} = \{ (i,co,succ) \mid (i,co,succ,pc^*,seg) \in M \} \qquad (5)$$

Cette dernière propriété nous permet d'étendre l'abréviation P'∈ PC(P), définie sur des points de M_0, aux points de M. Par la suite cette abréviation sera utilisée indifféremment pour les points de M_0 et de M.

Les propriétés suivantes décrivent l'ensemble PC* associé à chaque point P de M.

(6) spécifie que les éléments de PC* sont éléments de M_0 et que PC* est un ensemble (c'est-à-dire qu'il ne contient pas deux triplets identiques).

$$\forall P \in M \ \forall (P_1.i,P_1.co,P_1.succ) \in P.pc^* \qquad (6)$$
$$[\ (P_1.i,P_1.co,P_1.succ) \in M_0 \wedge$$
$$\forall (P_2.i,P_2.co,P_2.succ) \in P.pc^* - \{ (P_1.i,P_1.co,P_1.succ) \}$$
$$(P_1.i,P_1.co,P_1.succ) \neq (P_2.i,P_2.co,P_2.succ)$$
$$]$$

(7), (8) et (9) sont les conditions suffisantes d'appartenance à PC*(P):

$$\forall P \in M \quad (P.i,P.co,P.succ) \in P.pc^* \qquad (7)$$

$$\forall P \in M \ \forall P_1 \in M - \{P\} \qquad (8)$$
$$P_1 \in PC(P) \Rightarrow (P_1.i,P_1.co,P_1.succ) \in P.pc^*$$

$$\forall P \in M \ \forall P_1 \in M - \{P\} \ \forall P_2 \in M - \{P, P_1\} \quad (9)$$

$$(P_1.i, P_1.co, P_1.succ) \in P.pc^* \wedge P_2 \in PC(P_1)$$

$$\Rightarrow (P_2.i, P_2.co, P_2.succ) \in P.pc^*$$

Les conditions nécessaires pour qu'un point appartienne à PC*(P) sont données par (10) et (11). Les figures 9a et 9b expliquent la propriété (10) qui traite le cas où P et P_1 sont distants de moins de 25 pixels. Soit P et P_1 ne sont pas dans la même image (P_1 est candidat de P), soit P et P_1 sont dans la même image (P_1 n'est pas candidat de P) et alors il faut qu'il existe un point P_2 candidat à la fois de P et de P_1.

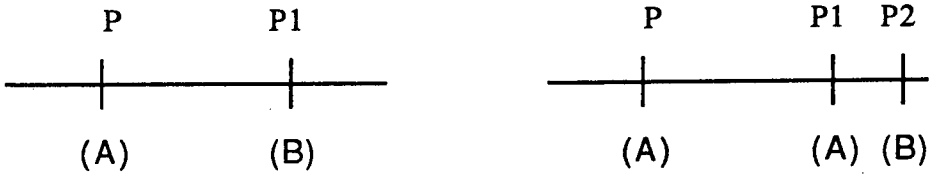

fig.9a et 9b propriété (10): P et P_1 distants de moins de 25 pixels.
(a) P et P_1 candidats; (b) P et P_1 non candidats

$$\forall P \in M \ \forall P_1 \in M - \{P\} \quad (10)$$

$$[\ P.co.y = P_1.co.y \quad \wedge \quad |\ P.co.x - P_1.co.x\ | \leq TF/2 \wedge$$

$$(P_1.i, P_1.co, P_1.succ) \in P.pc^*\ ]$$

$$\Rightarrow$$

$$[\ P.i \neq P_1.i \quad \vee \quad \exists P_2 \in M - \{P, P_1\} \ P_2 \in PC(P) \wedge P_2 \in PC(P_1)\ ]$$

Si par contre P et P_1 sont distants de plus de 25 pixels (11), pour que $(P_1.i, P_1.co, P_1.succ)$ appartienne à la fermeture P.pc*, il faut qu'il existe des points entre P et P_1 qui appartiennent aussi à P.pc*. Soit, (figure 10a), P_1 a un tel candidat P_2, soit, (figure 10b), P_1 a un candidat P_2 dont la distance avec P est supérieure à celle entre P et P_1 mais il existe un point P_3 candidat de P_2 (donc dans la même image que P_1) tel que la distance entre P_3 et P est inférieure à celle entre P et P_1. Comme cette propriété est vraie pour tous les points des deux images, par induction, on peut l'appliquer aux points P_2 et P_3 jusqu'à pouvoir leur appliquer (10) puisque la distance entre les points cherchés décroît strictement.

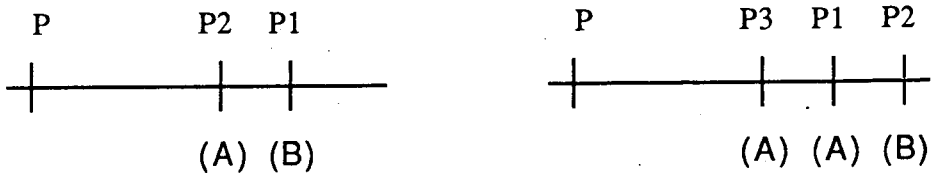

fig.10a et 10b propriété (11): P et P_1 distants de plus de 25 pixels

$$\forall P \in M \ \forall P_1 \in M - \{P\} \qquad (11)$$

$$[\ P.co.y = P_1.co.y \quad \wedge \quad |\ P.co.x - P_1.co.x\ | > TF/2 \wedge$$

$$(P_1.i, P_1.co, P_1.succ) \in P.pc^*\ ]$$

$$\Rightarrow$$

$$[\ \exists P_2 \in M - \{P, P_1\}$$

$$(P_2.i, P_2.co, P_2.succ) \in P.pc^* \wedge P_2 \in PC(P_1) \wedge$$

$$|\ P_2.co.x - P.co.x\ | < |\ P_1.co.x - P.co.x\ |$$

$$\vee$$

$$\exists P_2 \in M - \{P, P_1\}\ \exists P_3 \in M - \{P, P_1, P_2\}$$

$$|\ P_2.co.x - P.co.x\ | \geq |\ P_1.co.x - P.co.x\ | \wedge$$

$$(P_2.i, P_2.co, P_2.succ) \in P.pc^* \wedge P_2 \in PC(P_1) \wedge$$

$$(P_3.i, P_3.co, P_3.succ) \in P.pc^* \wedge P_3 \in PC(P_2) \wedge$$

$$|\ P_3.co.x - P.co.x\ | < |\ P_1.co.x - P.co.x\ |$$

$$]$$

(12) donne le domaine des valeurs que peut prendre le champ P.seg de P.

$$\forall P \in M \quad P.seg \in \{\ P.succ, NIL\ \} \qquad (12)$$

Un segment étant une sous-suite d'un vaisseau donné, on pose que P.seg = NIL lorsque P est le dernier point du segment auquel il appartient; sinon P est dans le même segment que son successeur de coordonnées P.succ, on pose alors P.seg = P.succ. Ainsi, le dernier segment d'un vaisseau se termine avec le dernier point de ce vaisseau. Il s'ensuit qu'un vaisseau réduit à un point forme un segment. La propriété (13) ne traite donc que le cas des vaisseaux ayant au moins deux points. Soient P_1 et P_2 deux points voisins, ils appartiennent au même segment si

et seulement si premièrement, ils ont même environnement de vaisseaux et deuxièmement, pour tous les couples de points (P'_1,P'_2) tels que P'_1 et P'_2 sont voisins, $(P'_1.i,P'_1.co,P'_1.succ) \in PC^*(P_1)$ et $(P'_2.i,P'_2.co,P'_2.succ) \in PC^*(P_2)$, P'_1 et P'_2 ont même environnement de vaisseaux. Par rapport à la présentation du paragraphe 4.2.1., il est à noter que (13) définit des propriétés communes aux points d'un même segment; les propriétés différenciant les points de deux segments différents s'obtiennent par contraposition.

$$
\begin{aligned}
&\forall P_1 \in M \ \forall P_2 \in M - \{ P_1 \} \qquad (13)\\
&\quad (P_1.i,P_1.succ) = (P_2.i,P_2.co) \Rightarrow\\
&\quad (\ P_1.seg = P_2.co \Leftrightarrow\\
&\qquad [\ EV(P_1,P_2) = EV(P_2,P_1) \wedge\\
&\qquad\quad [\ \forall P'_1 \in M - \{ P_1, P_2 \} \ \forall P'_2 \in M - \{ P_1, P_2, P'_1 \}\\
&\qquad\qquad [\ P'_1.i = P'_2.i \ \wedge\\
&\qquad\qquad\quad (\ P'_1.succ = P'_2.co \ \vee \ P'_2.succ = P'_1.co\ ) \wedge\\
&\qquad\qquad\quad (P'_1.i,P'_1.co,P'_1.succ) \in P_1.pc^* \wedge\\
&\qquad\qquad\quad (P'_2.i,P'_2.co,P'_2.succ) \in P_2.pc^*\\
&\qquad\qquad]\\
&\qquad\qquad \Rightarrow EV(P'_1,P'_2) = EV(P'_2,P'_1)\\
&\qquad\quad]\\
&\qquad]\\
&\quad)
\end{aligned}
$$

avec (paragraphe 4.2.1):

$$
\begin{aligned}
EV(P_1,P_2) = \{\ P'_1 \in M \mid P'_1 \in PC(P_1) \wedge [\ P'_1.succ = NIL \vee\\
[\ P'_1.succ.y \neq P_2.co.y \ \vee \ | P'_1.succ.x - P_2.co.x | > TF/2\ ]]\} \cup\\
\{\ P'_2 \in M \mid P'_2 \in PC(P_2) \wedge \exists P'_1 \in M \wedge P'_1 \in PC(P_1) \wedge P'_2.co = P'_1.succ\ \}
\end{aligned}
$$

Montrons maintenant comment cette spécification peut être utilisée pour dériver un programme GAMMA en appliquant la méthode décrite en 3.2.

4.3.2. Choix de l'état initial et de l'invariant.

Etape 1: Parmi les propriétés (1) à (13), on sait que (1) à (3) décrivent l'état M_0 donné par la squelettisation. (4) et (5) décrivent la correspondance entre M et M_0. (6) à (11) définissent les ensembles P.pc* et (12) les deux valeurs permises pour le champ P.seg. Les propriétés des graphes segmentés sont formalisées par (13). M est formé de quintuples dont les trois premiers champs proviennent de M_0 et dont les deux derniers champs P.pc* et P.seg doivent être calculés par le programme que nous dérivons. Leurs initialisations sont choisies au cours de la dérivation. Les ensembles P.pc* interviennent dans la propriété (13) donc la segmentation doit se dérouler en deux phases: la première calcule les ensembles P.pc* et la deuxième exploite ces ensembles pour former les segments.

A chaque phase correspond une décomposition des spécifications en un invariant et un variant; nous avons donc pour la première partie:

(I1) = (1) $\wedge$ (2) $\wedge$ (3) $\wedge$ (4) $\wedge$ (5) $\wedge$ (12)

(V1) = (6) $\wedge$ (7) $\wedge$ (8) $\wedge$ (9) $\wedge$ (10) $\wedge$ (11)

et pour la deuxième:

(I2) = (1) $\wedge$ (2) $\wedge$ (3) $\wedge$ (4) $\wedge$ (5) $\wedge$ (6) $\wedge$ (7) $\wedge$ (8) $\wedge$ (9) $\wedge$ (10) $\wedge$ (11) $\wedge$ (12)

(V2) = (13)

L'invariant initial (I1) est constitué des propriétés qu'on peut directement établir à partir de l'ensemble de départ M_0. Le premier programme établit (V1) qui est ensuite intégré dans l'invariant du deuxième programme. Nous dérivons le calcul des champs P.pc* puis celui des champs P.seg.

4.3.3. Dérivation du calcul des P.pc*.

Etape 1: Nous pouvons proposer deux initialisations faciles à réaliser par un programme GAMMA pour les champs P.pc*: un minorant, par exemple $\varnothing$, ou un majorant, par exemple l'ensemble des points qui sont sur la même ligne que P:

LIG(P) = { (P'.i,P'.co,P'.succ) | P' $\in$ M $\wedge$ P'.co.y = P.co.y }.

Le choix de $\varnothing$ revient à valider (6), (10) et (11) et donc à renforcer l'invariant (I1) en

(I'1) = (I1) $\wedge$ (6) $\wedge$ (10) $\wedge$ (11) tandis que le variant (V1) se réduit à (V'1) = (7) $\wedge$ (8) $\wedge$ (9). L'initialisation à LIG(P) valide (6), (7), (8) et (9) et donne la nouvelle décomposition des spécifications (I"1) = (I1) $\wedge$ (6) $\wedge$ (7) $\wedge$ (8) $\wedge$ (9) et (V"1) = (10) $\wedge$ (11). Ces deux initialisations conduisent à deux programmes duaux. Nous ne dérivons ici que celui qui correspond à la décomposition (I'1,V'1).

Etape 2: Les conditions de réaction sont dérivées à partir de la négation du variant. On a $\neg$ (V'1) = $\neg$ (7) $\vee$ $\neg$ (8) $\vee$ $\neg$ (9) soit:

$\neg$ (7) = $\exists$ P $\in$ M (P.i,P.co,P.succ) $\notin$ P.pc*

$\neg$ (8) = $\exists$ P $\in$ M $\exists$ P_1 $\in$ M – {P} $P_1 \in PC(P) \wedge (P_1.i,P_1.co,P_1.succ) \notin P.pc^*$

$\neg$ (9) = $\exists$ P $\in$ M $\exists$ P_1 $\in$ M – {P} $\exists P_2 \in M - \{P, P_1\}$

$(P_1.i,P_1.co,P_1.succ) \in P.pc^* \wedge P_2 \in PC(P_1) \wedge (P_2.i,P_2.co,P_2.succ) \notin P.pc^*$

qui se dérivent directement en:

Rpc7 (P) = (P.i,P.co,P.succ) $\notin$ P.pc*

Rpc8 $(P,P_1) = P_1 \in PC(P) \wedge (P_1.i,P_1.co,P_1.succ) \notin P.pc^*$

Rpc9 $(P,P_1,P_2) = (P_1.i,P_1.co,P_1.succ) \in P.pc^* \wedge P_2 \in PC(P_1) \wedge$

$(P_2.i,P_2.co,P_2.succ) \notin P.pc^*$

Etape 3: La fonction de terminaison standard $f(M) = N_R(M)$ ne permet pas de dériver les actions; en effet, pour Apc8, il est nécessaire d'ajouter $(P_1.i,P_1.co,P_1.succ)$ à P.pc* afin de rendre vraie la propriété (8) entre P et P_1. Or, l'introduction de $(P_1.i,P_1.co,P_1.succ)$ dans P.pc* rend vraies les conditions de réaction Rpc9 (P,P_1,P_2) pour tous les points P_2 candidats de P_1; f ne serait alors plus strictement décroissante. Nous essayons donc de faire croître strictement la fonction suivante:

$$f(M) = \sum_{P \in M} card(P.pc^*).$$

bornée supérieurement par:

$$\sum_{P \in M} card(LIG(P)).$$

Etape 4: Il faut maintenant déterminer les actions correspondantes aux conditions de réaction qui respectent l'invariant et font croître strictement f. Pour cela, il est nécessaire d'ajouter dans les P.pc* le triplet (P_1.i,P_1.co,P_1.succ) pour Apc8 et (P_2.i,P_2.co,P_2.succ) pour Apc9; de plus, afin de conserver la propriété (6) de l'invariant (P.pc* est un ensemble et tous ses éléments sont éléments de M_0), on ne peut rajouter que ces triplets dans P.pc*. Les seules actions possibles sont donc:

Apc7 (P) = { (P.i,P.co,P.succ,P.pc* $\cup$ {(P.i,P.co,P.succ)},P.seg) }

Apc8 (P,P_1) = { (P.i,P.co,P.succ,P.pc* $\cup$ {(P_1.i,P_1.co,P_1.succ)},P.seg),P_1 }

Apc9 (P,P_1,P_2) = { (P.i,P.co,P.succ,P.pc* $\cup$ {(P_2.i,P_2.co,P_2.succ)},P.seg),P_1,P_2 }

On obtient le programme suivant, dans lequel la notation M $\leftarrow$ N indique que le multi-ensemble M est initialisé au multi-ensemble N:

calcul_PC* (M) =

Γ ((Rpc7, Apc7), (Rpc8, Apc8), (Rpc9, Apc9))

(M $\leftarrow$ { (P.i, P.co, P.succ, $\varnothing$, P.seg) | (P.i, P.co, P.succ) $\in$ M0 })

avec

Rpc7 (P) = (P.i,P.co,P.succ) $\notin$ P.pc*

Apc7 (P) = {(P.i,P.co,P.succ,P.pc* $\cup$ { (P.i,P.co,P.succ) },P.seg) }

Rpc8 (P,P_1) = $P_1 \in$ PC(P) $\wedge$ (P_1.i,P_1.co,P_1.succ) $\notin$ P.pc*

Apc8 (P,P_1) = {(P.i,P.co,P.succ,P.pc*$\cup${(P_1.i,P_1.co,P_1.succ)},P.seg),P_1}

Rpc9 (P,P_1,P_2) =(P_1.i,P_1.co,P_1.succ) $\in$ P.pc* $\wedge$ $P_2 \in$ PC(P_1) $\wedge$
(P_2.i,P_2.co,P_2.succ) $\notin$ P.pc*

Apc9 (P,P_1,P_2) =

{(P.i,P.co,P.succ,P.pc* $\cup$ {(P_2.i,P_2.co,P_2.succ)},P.seg), P_1, P_2}

4.3.4. Dérivation du calcul des segments.

Etape 1: Rappelons que la décomposition des spécifications pour ce deuxième programme est la suivante:

$(I2) = (1) \wedge (2) \wedge (3) \wedge (4) \wedge (5) \wedge (6) \wedge (7) \wedge (8) \wedge (9) \wedge (10) \wedge (11) \wedge (12)$

$(V2) = (13)$

La propriété (12) nous suggère deux initialisations (P.seg=P.succ ou P.seg=NIL) et donc deux nouveaux variants et invariants. (13) s'écrit aussi (13') $\wedge$ (13") $\wedge$ (13"') en utilisant l'équivalence:

$[A \Rightarrow (B \Leftrightarrow (C_1 \wedge C_2))] \equiv [((A \wedge B) \Rightarrow C_1) \wedge ((A \wedge B) \Rightarrow C_2) \wedge ((A \wedge C_1 \wedge C_2) \Rightarrow B)]$

avec:

$A \equiv (P_1.i, P_1.succ) = (P_2.i, P_2.co)$

$B \equiv P_1.seg = P_2.co$

$C_1 \equiv EV(P_1,P_2) = EV(P_2,P_1)$

$C_2 \equiv [\ \forall P'_1 \in M - \{ P_1, P_2 \}\ \forall P'_2 \in M - \{ P_1, P_2, P'_1 \}$

$[\ P'_1.i = P'_2.i \wedge$

$(P'_1.succ = P'_2.co \vee P'_2.succ = P'_1.co) \wedge$

$(P'_1.i, P'_1.co, P'_1.succ) \in P_1.pc^* \wedge$

$(P'_2.i, P'_2.co, P'_2.succ) \in P_2.pc^*$

$]$

$\Rightarrow EV(P'_1,P'_2) = EV(P'_2,P'_1)$

$]$

L'intérêt de cette transformation est de mettre le variant sous la forme de conjonctions. C'est une stratégie fructueuse en général car une partie des conjoints peut souvent être intégrée à l'invariant, ce qui affaiblit d'autant le variant. On a donc, après avoir mis les formules sous forme prénexe:

$\forall P_1 \in M \qquad \forall P_2 \in M - \{ P_1 \}$ (13')

$(P_1.i, P_1.succ, P_1.seg) = (P_2.i, P_2.co, P_2.co) \Rightarrow EV(P_1,P_2) = EV(P_2,P_1)$

$$\forall P_1 \in M \qquad \forall P_2 \in M - \{ P_1 \} \tag{13''}$$

$$\forall P'_1 \in M - \{ P_1, P_2 \} \qquad \forall P'_2 \in M - \{ P_1, P_2, P'_1 \}$$

$$[\quad (P_1.i, P_1.succ, P_1.seg) = (P_2.i, P_2.co, P_2.co) \wedge$$

$$P'_1.i = P'_2.i \wedge$$

$$(P'_1.succ = P'_2.co \vee P'_2.succ = P'_1.co) \wedge$$

$$(P'_1.i, P'_1.co, P'_1.succ) \in P_1.pc^* \wedge$$

$$(P'_2.i, P'_2.co, P'_2.succ) \in P_2.pc^*$$

$$] \Rightarrow EV(P'_1, P'_2) = EV(P'_2, P'_1)$$

$$\forall P_1 \in M \ \forall P_2 \in M - \{ P_1 \} \tag{13'''}$$

$$[(P_1.i, P_1.succ) = (P_2.i, P_2.co) \wedge EV(P_1, P_2) = EV(P_2, P_1) \wedge$$

$$\forall P'_1 \in M - \{ P_1, P_2 \} \ \forall P'_2 \in M - \{ P_1, P_2, P'_1 \}$$

$$[P'_1.i = P'_2.i \wedge$$

$$(P'_1.succ = P'_2.co \vee P'_2.succ = P'_1.co) \wedge$$

$$(P'_1.i, P'_1.co, P'_1.succ) \in P_1.pc^* \wedge$$

$$(P'_2.i, P'_2.co, P'_2.succ) \in P_2.pc^*$$

$$] \Rightarrow EV(P'_1, P'_2) = EV(P'_2, P'_1)$$

$$] \Rightarrow P_1.seg = P_2.co$$

Pour chaque point P, si nous choisissons comme état initial P.seg = P.succ, (13''') est validée et nous pouvons renforcer l'invariant en (I'2) = (I2) $\wedge$ (13''') et affaiblir le variant en (V'2) = (13') $\wedge$ (13''). Par contre, si nous initialisons P.seg à NIL, (13') et (13'') sont validées et nous regroupons les spécifications en (I''2) = (I2) $\wedge$ (13') $\wedge$ (13'') et (V''2) = (13'''). Nous dérivons d'abord le programme correspondant à la décomposition (I'2,V'2).

Etape 2: L'évaluation des ensembles $EV(P_1,P_2)$, tels que nous les avons définis, fait intervenir l'état global M; ceci est incompatible avec la propriété de localité de GAMMA. Cependant, la propriété (8) de l'invariant:

$$\forall P \in M \ \forall P_1 \in M - \{P\} \ \ P_1 \in PC(P) \Rightarrow (P_1.i, P_1.co, P_1.succ) \in P.pc^*$$

et la propriété $P' \in M \wedge P' \in PC(P)$ permettent d'inférer:

$(P'.i,P'.co,P'.succ) \in P.pc^* \wedge (P'.i,P'.co,P'.succ) \in PC(P)$

qui peut être calculé localement en connaissant P. Nous posons donc:

$EV(P_1,P_2) =$

$\{ (P'_1.i, P'_1.co, P'_1.succ) \in P1.pc^* \mid (P'_1.i, P'_1.co, P'_1.succ) \in PC(P_1) \wedge$

$[P'_1.succ = NIL \vee [P'_1.succ.y \neq P_2.co.y \vee | P'_1.succ.x - P_2.co.x | > TF/2]] \}$

$\cup \{ (P'_2.i, P'_2.co, P'_2.succ) \in P2.pc^* \mid (P'_2.i, P'_2.co, P'_2.succ) \in PC(P_2) \wedge$

$\exists (P'_1.i, P'_1.co, P'_1.succ) \in P1.pc^* \wedge (P'_1.i, P'_1.co, P'_1.succ) \in PC(P_1)$

$\wedge P'_2.co = P'_1.succ \}$

Les ensembles EV qui étaient composés de quintuples sont désormais des ensembles de triplets mais ceci n'a aucune incidence sur les prédicats qui font intervenir des comparaisons d'environnements de vaisseaux. Nous dérivons les conditions de réaction directement à partir de la négation du variant et nous obtenons:

Rseg1 $(P_1,P_2) =$

$(P_1.i,P_1.succ,P_1.seg) = (P_2.i,P_2.co,P_2.co) \wedge EV(P_1,P_2) \neq EV(P_2,P_1)$

Rseg2 $(P_1,P_2,P'_1,P'_2) =$

$(P_1.i,P_1.succ,P_1.seg) = (P_2.i,P_2.co,P_2.co) \wedge P'_1.i = P'_2.i \wedge$

$(P'_1.succ = P'_2.co \vee P'_2.succ = P'_1.co) \wedge$

$(P'_1.i,P'_1.co,P'_1.succ) \in P_1.pc^* \wedge (P'_2.i,P'_2.co,P'_2.succ) \in P_2.pc^* \wedge$

$EV(P'_1,P'_2) \neq EV(P'_2,P'_1)$

Les ensembles EV intervenant dans les conditions de réaction peuvent être calculés par des programmes GAMMA.

Etape 3: Nous essayons comme fonction de terminaison la fonction standard $f(M) = N_R(M)$, nombre de n-uplets pouvant réagir.

Etape 4: D'après l'invariant (I'2), le nombre de points dans M est constant et seul le champ P.seg des points P de M est susceptible d'être modifié; d'autre part, (12) impose au plus deux valeurs distinctes pour P.seg. Par conséquent, il faut changer la valeur P.seg d'au moins l'un des points réagissant. L'unique possibilité pour l'action Aseg1 est d'affecter à P_1.seg la valeur NIL; cette dernière est possible puisque $P_2.co \neq NIL$: la fonction de

terminaison décroît donc strictement. Ce qui nous donne l'action suivante:

Aseg1 (P_1,P_2) = { $(P_1.i,P_1.co,P_1.succ,P_1.pc^*,NIL)$, P_2 }

Pour Aseg2, on obtient de la même manière:

Aseg2 (P_1,P_2,P'_1,P'_2) = { $(P_1.i,P_1.co,P_1.succ,P_1.pc^*,NIL)$, P_2, P'_1, P'_2 }

Nous avons dérivé le programme suivant:

segmentation (M_0) =

Γ **((Rseg1,Aseg1),(Rseg2,Aseg2))**

(M ← { (P.i,P.co,P.succ,P.pc*,P.succ) |

(P.i,P.co,P.succ,P.pc*,P.seg) ∈ Calcul_PC* (M0) })

avec

Rseg1 (P_1,P_2) =

$(P_1.i,P_1.succ,P_1.seg) = (P_2.i,P_2.co,P_2.co) \wedge EV(P_1,P_2) \neq EV(P_2,P_1)$

Aseg1 (P_1,P_2) = { $(P_1.i,P_1.co,P_1.succ,P_1.pc^*,NIL)$, P_2 }

Rseg2 (P_1,P_2,P'_1,P'_2) =

$(P_1.i,P_1.succ,P_1.seg) = (P_2.i,P_2.co,P_2.co) \wedge P'_1.i = P'_2.i \wedge$

$(P'_1.succ = P'_2.co \vee P'_2.succ = P'_1.co) \wedge$

$(P'_1.i,P'_1.co,P'_1.succ) \in P_1.pc^* \wedge (P'_2.i,P'_2.co,P'_2.succ) \in P_2.pc^* \wedge$

$EV(P'_1,P'_2) \neq EV(P'_2,P'_1)$

Aseg2 (P_1,P_2,P'_1,P'_2) = {$(P_1.i,P_1.co,P_1.succ,P_1.pc^*,NIL)$, P_2, P'_1, P'_2 }

Nous donnons maintenant la dérivation correspondant à la décomposition (I"2,V"2).

Etape 2: Dans ce cas, la méthode de dérivation rencontre une difficulté car la négation de (13'") inclut une propriété (14) qui contient des quantificateurs universels portant sur l'état global M:

$\forall P'_1 \in M - \{ P_1, P_2 \} \forall P'_2 \in M - \{ P_1, P_2, P'_1 \}$ (14)

$[P'_1.i = P'_2.i \wedge (P'_1.succ = P'_2.co \vee P'_2.succ = P'_1.co) \wedge$

$(P'_1.i,P'_1.co,P'_1.succ) \in P_1.pc^* \wedge (P'_2.i,P'_2.co,P'_2.succ) \in P_2.pc^*]$

$\Rightarrow EV(P'_1,P'_2) = EV(P'_2,P'_1)$

Cette propriété peut se localiser de la façon suivante: comme $(P'_1.i,P'_1.co,P'_1.succ) \in P_1.pc^*$ et $(P'_2.i,P'_2.co,P'_2.succ) \in P_2.pc^*$, nous transformons:

$\forall\ P'_1 \in M - \{ P_1, P_2 \}$ et $\forall\ P'_2 \in M - \{ P_1, P_2, P'_1 \}$

en

$\forall\ (P'_1.i,P'_1.co,P'_1.succ) \in P_1.pc^* - \{ (P_1.i,P_1.co,P_1.succ),\ (P_2.i,P_2.co,P_2.succ) \}$ et

$\forall\ (P'_2.i,P'_2.co,P'_2.succ) \in P_2.pc^* - \{ (P_1.i,P_1.co,P_1.succ),\ (P_2.i,P_2.co,P_2.succ),$
$(P'_1.i,P'_1.co,P'_1.succ) \}$

Mais les environnements de vaisseaux $EV(P'_1,P'_2)$ et $EV(P'_2,P'_1)$ se calculent à partir des ensembles $P'_1.pc^*$ et $P'_2.pc^*$ dont nous ne disposons plus à la suite de cette transformation. Cette difficulté est résolue grâce à la propriété suivante:

$\forall\ P \in M\ \forall\ P' \in M - \{ P \}\quad (P'.i,P'.co,P'.succ) \in P.pc^* \Rightarrow P'.pc^* = P.pc^*$

car PC* est la fermeture réflexive et transitive de la relation PC. D'où, dans (13'''):

$P'_1.pc^* = P_1.pc^*$ et $P'_2.pc^* = P_2.pc^*$. Nous définissons par conséquent l'environnement de vaisseaux de $(P'_1.i,P'_1.co,P'_1.succ) \in P_1.pc^*$ par rapport à $(P'_2.i,P'_2.co,P'_2.succ) \in P_2.pc^*$ par:

$EV'(\ (P'_1.i,P'_1.co,P'_1.succ),\ P_1.pc^*,\ (P'_2.i,P'_2.co,P'_2.succ),\ P_2.pc^*) =$

$\{\ (\ P''_1.i,\ P''_1.co,\ P''_1.succ) \in P1.pc^* \mid$

$(\ P''_1.i,\ P''_1.co,\ P''_1.succ) \in PC(\ (P'_1.i,P'_1.co,P'_1.succ)\ ) \wedge$

$[\ P''_1.succ = NIL \vee$

$[\ P''_1.succ.y \neq P'_2.co.y \ \vee\ |\ P''_1.succ.x - P'_2.co.x\ | > TF/2\ ]]\ \}$

$\cup\ \{\ (\ P''_2.i,\ P''_2.co,\ P''_2.succ) \in P2.pc^* \mid$

$(\ P''_2.i,\ P''_2.co,\ P''_2.succ) \in PC(\ (P'_2.i,P'_2.co,P'_2.succ)\ ) \wedge$

$\exists\ (\ P''_1.i,\ P''_1.co,\ P''_1.succ) \in P_1.pc^* \wedge$

$(\ P''_1.i,\ P''_1.co,\ P''_1.succ) \in PC(\ (P'_1.i,P'_1.co,P'_1.succ)\ )$

$\wedge\ P''_2.co = P''_1.succ\ \}$

Finalement, nous obtenons une formulation locale de (14):

$\forall$ (P'_1.i,P'_1.co,P'_1.succ) $\in$ P_1.pc* – { (P_1.i,P_1.co,P_1.succ), (P_2.i,P_2.co,P_2.succ) }

$\forall$ (P'_2.i,P'_2.co,P'_2.succ) $\in$ P_2.pc* – { (P_1.i,P_1.co,P_1.succ), (P_2.i,P_2.co,P_2.succ), (P'_1.i,P'_1.co,P'_1.succ) }

[P'_1.i = P'_2.i $\wedge$ (P'_1.succ = P'_2.co $\vee$ P'_2.succ = P'_1.co) $\wedge$

(P'_1.i,P'_1.co,P'_1.succ) $\in$ P_1.pc* $\wedge$ (P'_2.i,P'_2.co,P'_2.succ) $\in$ P_2.pc*

] $\Rightarrow$ EV'((P'_1.i,P'_1.co,P'_1.succ), P_1.pc*, (P'_2.i,P'_2.co,P'_2.succ), P_2.pc*) =

EV'((P'_2.i,P'_2.co,P'_2.succ), P_2.pc*, (P'_1.i,P'_1.co,P'_1.succ), P_1.pc*)

équivalente à:

{ (P'_1.i,P'_1.co,P'_1.succ) $\in$ P_1.pc* – { (P_1.i,P_1.co,P_1.succ), (P_2.i,P_2.co,P_2.succ) } |

$\exists$ (P'_2.i,P'_2.co,P'_2.succ) $\in$ P_2.pc* – { (P_1.i,P_1.co,P_1.succ), (P_2.i,P_2.co,P_2.succ), (P'_1.i,P'_1.co,P'_1.succ) }

P'_1.i = P'_2.i $\wedge$ (P'_1.succ = P'_2.co $\vee$ P'_2.succ = P'_1.co) $\wedge$

(P'_1.i,P'_1.co,P'_1.succ) $\in$ P_1.pc* $\wedge$ (P'_2.i,P'_2.co,P'_2.succ) $\in$ P_2.pc* $\wedge$

EV'((P'_1.i,P'_1.co,P'_1.succ), P_1.pc*, (P'_2.i,P'_2.co,P'_2.succ), P_2.pc*) $\neq$

EV'((P'_2.i,P'_2.co,P'_2.succ), P_2.pc*, (P'_1.i,P'_1.co,P'_1.succ), P_1.pc*)} = $\varnothing$

Nous avons donc dérivé la condition de réaction:

Rseg3 (P_1,P_2) =

(P_1.i,P_1.succ) = (P_2.i,P_2.co) $\wedge$ EV(P_1,P_2) = EV(P_2,P_1) $\wedge$

{ (P'_1.i,P'_1.co,P'_1.succ) $\in$ P_1.pc* – {(P_1.i,P_1.co,P_1.succ), (P_2.i,P_2.co,P_2.succ) } |

$\exists$ (P'_2.i,P'_2.co,P'_2.succ) $\in$ P_2.pc* – { (P_1.i,P_1.co,P_1.succ), (P_2.i,P_2.co,P_2.succ), (P'_1.i,P'_1.co,P'_1.succ)}

P'_1.i = P'_2.i $\wedge$ (P'_1.succ = P'_2.co $\vee$ P'_2.succ = P'_1.co) $\wedge$

(P'_1.i,P'_1.co,P'_1.succ) $\in$ P_1.pc* $\wedge$ (P'_2.i,P'_2.co,P'_2.succ) $\in$ P_2.pc* $\wedge$

EV'((P'_1.i,P'_1.co,P'_1.succ), P_1.pc*, (P'_2.i,P'_2.co,P'_2.succ), P_2.pc*) $\neq$

EV'((P'_2.i,P'_2.co,P'_2.succ), P_2.pc*, (P'_1.i,P'_1.co,P'_1.succ), P_1.pc*)} = $\varnothing$

$\wedge$ P_1.seg $\neq$ P_2.co

Etape 3: Nous essayons comme fonction de terminaison la fonction standard f(M) = N_R(M).

Etape 4: Pour faire décroître cette fonction, il est nécessaire de former des segments plus importants en affectant P1.succ aux points P1 réagissant. Nous obtenons donc Aseg3:

Aseg3 (P_1,P_2) = { $(P_1.i,P_1.co,P_1.succ,P_1.pc^*,P_1.succ)$, P_2 }

et le programme:

segmentation' (M0) =

Γ (Rseg3,Aseg3)

(M ← { (P.i,P.co,P.succ,P.pc*,NIL) |

(P.i,P.co,P.succ,P.pc*,P.seg) ∈ Calcul_PC* (M0) })

avec

Rseg3 $(P_1,P_2) = (P_1.i,P_1.succ) = (P_2.i,P_2.co) \wedge EV(P_1,P_2) = EV(P_2,P_1) \wedge$

$\{(P'_1.i,P'_1.co,P'_1.succ) \in P_1.pc^* - \{(P_1.i,P_1.co,P_1.succ),$

$(P_2.i,P_2.co,P_2.succ)\} \mid$

$\exists\ (P'_2.i,P'_2.co,P'_2.succ) \in P_2.pc^* - \{ (P_1.i,P_1.co,P_1.succ),$

$(P_2.i,P_2.co,P_2.succ),\ (P'_1.i,P'_1.co,P'_1.succ)\}$

$P'_1.i = P'_2.i \wedge (P'_1.succ = P'_2.co \vee P'_2.succ = P'_1.co) \wedge$

$(P'_1.i,P'_1.co,P'_1.succ) \in P_1.pc^* \wedge (P'_2.i,P'_2.co,P'_2.succ) \in P_2.pc^* \wedge$

$EV'((P'_1.i,P'_1.co,P'_1.succ),P_1.pc^*,(P'_2.i,P'_2.co,P'_2.succ),P_2.pc^*) \neq$

$EV'((P'_2.i,P'_2.co,P'_2.succ),P_2.pc^*,(P'_1.i,P'_1.co,P'_1.succ),P_1.pc^*)$

$\} = \varnothing$

$\wedge\ P_1.seg \neq P_2.co$

Aseg3 $(P_1,P_2) = \{ (P_1.i,P_1.co,P_1.succ,P_1.pc^*,P_1.succ),\ P_2 \}$

Nous avons dérivé deux programmes de segmentation à partir des spécifications (1) à (13). Les deux programmes résolvent le même problème et sont parfaitement duaux. Le premier part d'un état initial M_0 correspondant à l'absence de segmentation. Les segments sont constitués de vaisseaux entiers qui sont "découpés" par le programme. En revanche, l'état initial du programme dual correspond à une segmentation maximum: chaque point constitue un segment. Au cours du déroulement de ce deuxième algorithme, le chaînage est

progressivement rétabli entre les points devant faire partie d'un même segment.

Ainsi, malgré l'aspect assez ardu du problème et une fois les spécifications écrites, il a été possible de dériver des algorithmes dont la correction est prouvée. De plus, ceux-ci peuvent être mis en oeuvre de façon très parallèle, comme nous le montrons dans la partie suivante.

5. Mise en œuvre de GAMMA.

Lors de l'exécution d'un programme GAMMA, il faut construire tous les n-uplets qui vérifient la condition de réaction. Un modèle d'évaluation possible consiste à former tous les n-uplets du multi-ensemble de façon à appliquer la condition de réaction sur chaque n-uplet, et éventuellement l'action. Nous présentons en premier lieu une mise en œuvre distribuée de ce modèle d'évaluation; nous montrons ensuite comment on peut l'optimiser en limitant le nombre de n-uplets à former.

5.1. Mise en œuvre de GAMMA sur un réseau de processeurs.

L'aspect local des opérations sur les données autorise un parallélisme important: le calcul de la condition de réaction et l'application de l'action sur un n-uplet d'éléments s'effectuent indépendamment du reste des éléments; plusieurs actions sur des n-uplets disjoints peuvent donc s'effectuer en parallèle. Dans la réalisation d'une mise en œuvre parallèle, plusieurs choix sont possibles [Banâtre 87, 88]:

(1) Le premier choix concerne la distribution des valeurs sur le réseau de processeurs. Nous avons exclu la duplication des valeurs qui aurait entraîné des problèmes importants de cohérence de données et une perte de la localité. Les valeurs sont donc réparties sans duplication sur les processeurs.

(2) Le deuxième choix concerne le processus de recherche des n-uplets. Ce processus doit vérifier les propriétés suivantes:

- absence d'interblocage,
- formation de tous les n-uplets possibles afin que toutes les réactions soient effectuées,
- détection de la terminaison: le système doit s'arrêter quand plus aucune condition de

réaction n'est vérifiée.

Pour assurer que tous les n-uplets sont réalisés, deux solutions sont possibles:

- on peut utiliser un réseau d'intercommunication où les valeurs sont immobiles; il faut alors trouver un protocole de communication assurant que chaque processeur communique avec tous les autres, de façon à confronter toutes les valeurs.

- on peut considérer que les valeurs se déplacent sur le réseau; il faut alors trouver un protocole qui gère le déplacement des valeurs afin qu'elles se rencontrent toutes en un temps fini.

(3) Le troisième choix concerne le contrôle du système; on peut opter pour un contrôle centralisé: un processeur particulier a une vue globale sur le réseau et il est responsable du protocole de communication et de la terminaison. Une deuxième solution est d'utiliser un contrôle distribué où aucun processeur n'a de vue globale sur le système; chaque processeur contribue alors au protocole de communication et à la détection de la terminaison.

Une machine synchrone à contrôle centralisé basée sur un réseau d'intercommunication est décrite dans [Banâtre 88]; nous présentons ici une machine asynchrone à contrôle distribué, qui a été mise en œuvre sur un iPSC/2 doté de 64 processeurs [Creveuil 87]. Cette machine vise à l'évaluation parallèle de GAMMA dans le cas de réactions binaires, c'est-à-dire agissant sur des couples d'éléments. De plus, on suppose que les réactions produisent exactement deux éléments et que l'on dispose d'autant de processeurs que de valeurs. Ces hypothèses peuvent être levées en gardant le même schéma de mise en œuvre.

La solution proposée consiste à répartir les valeurs du multi-ensemble sur une chaîne de N processeurs, à raison d'une valeur par processeur. Il n'y a pas de mémoire partagée et chaque processeur ne connait que ses voisins dans la chaîne. Afin d'assurer que tous les couples sont réalisés, les valeurs doivent se déplacer sur le réseau. On leur attribue pour cela une direction: chaque valeur se déplace dans le sens de sa direction et la direction change de sens à chaque extrémité de la chaîne. Un processeur ne cherche à communiquer qu'avec le voisin qui est dans la direction de sa valeur et la confrontation entre les valeurs n'a lieu que si

les deux valeurs ont des directions opposées. Ainsi, tous les couples sont réalisés et il n'y a pas de risque d'interblocage. De plus on prouve qu'en N-1 échanges consécutifs, une valeur rencontre N-1 valeurs différentes.

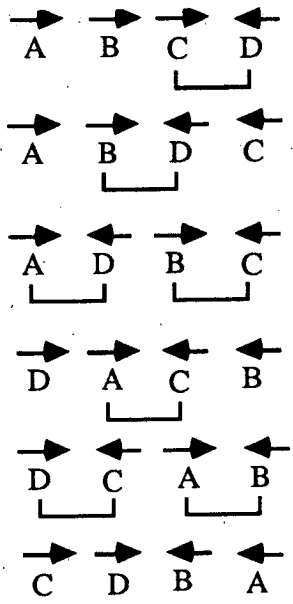

fig.11 Réalisation de tous les couples avec 4 processeurs. Les flèches représentent les directions associées aux valeurs. Un lien entre deux processeurs dénote que ceux-ci communiquent.

Enfin, chaque processeur est capable de détecter localement la terminaison en tenant compte du nombre d'échanges consécutifs sans réaction effectués par chaque valeur.

La mise en œuvre réalisée sur l'iPSC/2 a permis de mesurer le degré de parallélisme obtenu avec un tel schéma d'exécution. Précisons cependant que le rôle de cette mise en œuvre était principalement d'exhiber le parallélisme; pour obtenir une implémentation réellement efficace, il faudrait diminuer la combinatoire impliquée par la définition du formalisme lui-même; nous y revenons plus loin. Les résultats temporels obtenus sur quelques exemples montrent que, jusqu'à un certain seuil, le gain en rapidité augmente avec le nombre de processeurs; l'overhead engendré par les communications augmente peu. De plus, un outil graphique permettant de visualiser l'exécution parallèle des programmes a été développé. Il laisse apparaître qu'il est rare que de nombreux processeurs soient bloqués, en attente d'une communication avec un de leurs voisins; les processeurs travaillent donc effectivement en

parallèle la plupart du temps, parce que le réseau s'équilibre naturellement.

5.2. Optimisation du modèle d'évaluation.

Lorsqu'on forme tous les n-uplets, le nombre minimum de conditions de réaction à évaluer est de l'ordre du cardinal du multi-ensemble argument à la puissance de l'arité maximale des conditions de réaction. En effet, si N est le cardinal du multi-ensemble et n la plus grande arité des conditions de réaction, il y a A(N,n) n-uplets différents (avec A(N,n)=(N!)/(N-n)!). Or, il est possible d'éviter un certain nombre d'évaluations de conditions de réaction par analyse statique des programmes. Lorsque des propriétés invariantes apparaissent dans les conditions de réaction, nous pouvons déduire des relations entre les valeurs; ceci nous permet de déduire une topologie des valeurs, c'est-à-dire des processeurs dans l'hypothèse où l'on a une valeur par processeur. Les valeurs ne se déplacent plus sur le réseau, chaque processeur connaissant les processeurs avec lesquels il a à communiquer [Moguérou 88].

Prenons l'exemple du programme de calcul des PC^*, dérivé au chapitre 4:

calcul_PC* (M) =

Γ ((Rpc7, Apc7), (Rpc8, Apc8), (Rpc9, Apc9))

(M ← { (P.i, P.co, P.succ, ∅, P.seg) | (P.i, P.co, P.succ) ∈ M0 })

avec

Rpc7 (P) = (P.i,P.co,P.succ) ∉ P.pc*

Apc7 (P) = {(P.i,P.co,P.succ,P.pc* ∪ { (P.i,P.co,P.succ) },P.seg) }

Rpc8 (P,P_1) = P_1 ∈ PC(P) ∧ (P_1.i,P_1.co,P_1.succ) ∉ P.pc*

Apc8 (P,P_1) = {(P.i,P.co,P.succ,P.pc*∪{(P_1.i,P_1.co,P_1.succ)},P.seg),P_1}

Rpc9 (P,P_1,P_2) = (P_1.i,P_1.co,P_1.succ) ∈ P.pc* ∧ P_2 ∈ PC(P_1) ∧
(P_2.i,P_2.co,P_2.succ) ∉ P.pc*

Apc9 (P,P_1,P_2) =

{(P.i,P.co,P.succ,P.pc* ∪ {(P_2.i,P_2.co,P_2.succ)},P.seg), P_1, P_2}

Comme Rpc9 est d'arité 3, ce programme est au moins en $O(n^3)$. Détaillons sur chaque

condition de réaction comment il est possible de diminuer le nombre de n-uplets à former.

La propriété (5) ($\{(i,co,succ) \in M_0\} = \{(i,co,succ) \mid (i,co,succ,pc^*,seg) \in M\}$) faisant partie de l'invariant, le prédicat $P_1 \in PC(P)$ $(P_1.i \neq P.i \wedge P_1.co.y = P.co.y \wedge |P_1.co.x - P.co.x| \leq TF/2)$ qui apparait dans Rpc8 est invariant. Nous pouvons donc obtenir les informations suivantes:

(1) De $P_1.co.y=P.co.y$ nous déduisons un partitionnement des processeurs en ligne, chaque ligne de processeurs représentant une ligne de l'image. Chaque processeur ne communique qu'avec les processeurs de sa ligne (figure 12).

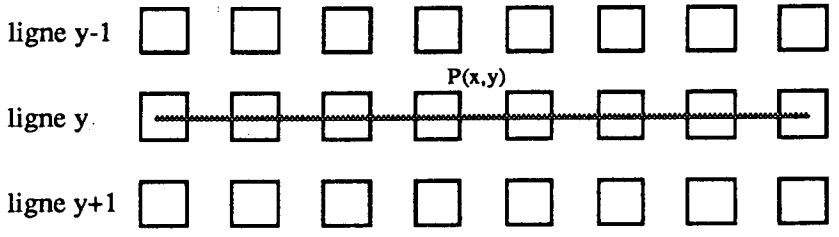

fig.12 Topologie déduite de $P_1.co.y=P.co.y$. Le trait grisé représente l'ensemble des processeurs avec lesquels P(x,y) peut communiquer.

(2) Comme $P_1.i \neq P.i$, ne réagissent entre eux que des points d'images différentes. On peut scinder chaque ligne de processeurs en deux classes, chaque classe possédant les points d'une image. On se limite alors à des communications inter-classes (figure 13).

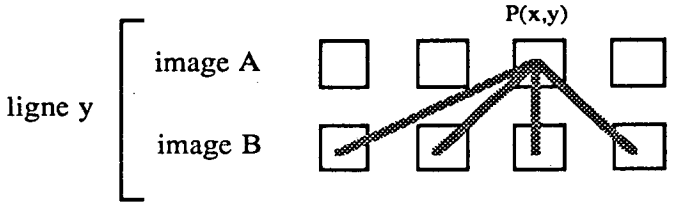

fig.13 Topologie déduite de $P_1.i \neq P.i \wedge P_1.co.y=P.co.y$

En revanche, on ne peut pas définir statiquement les liaisons entre processeurs telles que $|P_1.co.x - P.co.x| \leq TF/2$ soit vérifié pour chaque couple de points (P, P_1).

Dans le cas de Rpc9, comme $P_2 \in PC(P_1)$, on applique à P_2 et P_1 le même raisonnement qu'à P et P_1 dans le cas de Rpc8; ceci nous donne les relations entre les processeurs contenant P_1 et P_2. De plus, comme par construction tous les points de $P.pc^*$

sont sur la même ligne que P, on sait que P et P_1 sont sur la même ligne (par $(P_1.i,P_1.co,P_1.succ) \in P.pc^*$) et donc le processeur contenant P peut être n'importe quel processeur des deux classes (A ou B) de la ligne (figure 14).

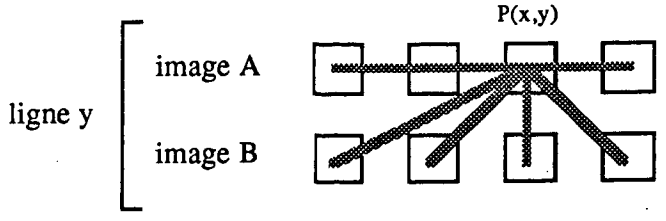

fig.14 Topologie pour l'évaluation de Rpc9; le processeur représentant P peut communiquer avec n'importe quel processeur de la ligne y, alors que les processeurs contenant les points P_1 et P_2 se trouvent respectivement sur chacune des deux classes.

Le réseau final est obtenu en faisant la superposition des topologies obtenues pour les deux conditions de réaction; on obtient dans ce cas le réseau de la figure 14.

Analysons de la même façon le programme de segmentation suivant:

segmentation (M_0) =

Γ **((Rseg1,Aseg1),(Rseg2,Aseg2))**

(M ← { (P.i,P.co,P.succ,P.pc*,P.succ) |

(P.i,P.co,P.succ,P.pc*,P.seg) ∈ Calcul_PC* (M0) })

avec

Rseg1 (P_1,P_2) =

$(P_1.i,P_1.succ,P_1.seg) = (P_2.i,P_2.co,P_2.co) \wedge EV(P_1,P_2) \neq EV(P_2,P_1)$

Aseg1 $(P_1,P_2) = \{ (P_1.i,P_1.co,P_1.succ,P_1.pc^*,NIL),\ P_2 \}$

Rseg2 (P_1,P_2,P'_1,P'_2) =

$(P_1.i,P_1.succ,P_1.seg) = (P_2.i,P_2.co,P_2.co) \wedge P'_1.i = P'_2.i \wedge$

$(P'_1.succ = P'_2.co \vee P'_2.succ = P'_1.co) \wedge$

$(P'_1.i,P'_1.co,P'_1.succ) \in P_1.pc^* \wedge (P'_2.i,P'_2.co,P'_2.succ) \in P_2.pc^*$

$\wedge EV(P'_1,P'_2) \neq EV(P'_2,P'_1)$

Aseg2 $(P_1,P_2,P'_1,P'_2) = \{(P_1.i,P_1.co,P_1.succ,P_1.pc^*,NIL),\ P_2,\ P'_1,\ P'_2 \}$

D'aprés la propriété (5), le prédicat (P_1.i,P_1.succ)=(P_2.i,P_2.co) apparaissant dans Rseg1 est invariant; pour chaque processeur, on limite donc à deux le nombre de processeurs avec lesquels il communique (en tenant compte du sens de parcours, figure 15).

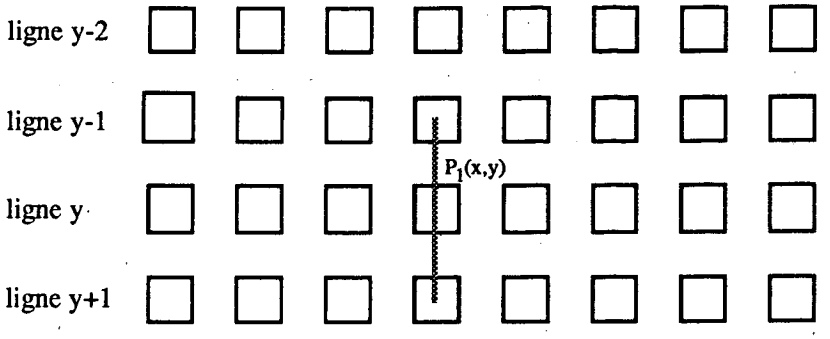

fig.15 Topologie déduite de (P_1.i,P_1.succ)=(P_2.i,P_2.co). Le processeur représentant P_1(x,y) peut communiquer avec le processeur possédant le successeur de P_1 qui se trouve soit sur la ligne y-1, soit sur la ligne y+1, suivant le sens de parcours; chaque colonne représente donc les vaisseaux de l'image.

Pour Rseg2, il faut définir une topologie qui permette la communication entre quatre processeurs. Les prédicats invariants (P_1.i,P_1.succ)=(P_2.i,P_2.co) et [(P'_1.i=P'_2.i) $\wedge$ (P'_1.succ=P'_2.co $\vee$ P'_2.succ=P'_1.co)] définissent les liaisons entre P_1 et P_2 et entre P'_1 et P'_2 de la même façon que pour Rseg1 (figure 15). De plus, comme les ensembles P.pc* sont invariants pour chaque point P, les prédicats (P'_1.i,P'_1.co,P'_1.succ)$\in$ P_1.pc* et (P'_2.i,P'_2.co,P'_2.succ)$\in$ P_2.pc* sont invariants; ils définissent les liaisons entre P_1 et P'_1 et entre P_2 et P'_2. Puisque l'ensemble P.pc* est inclus dans l'ensemble des points appartenant à la même ligne que P, on étend les relations de chaque processeur à tous les processeurs de la ligne. Nous obtenons alors la topologie de la figure 16.

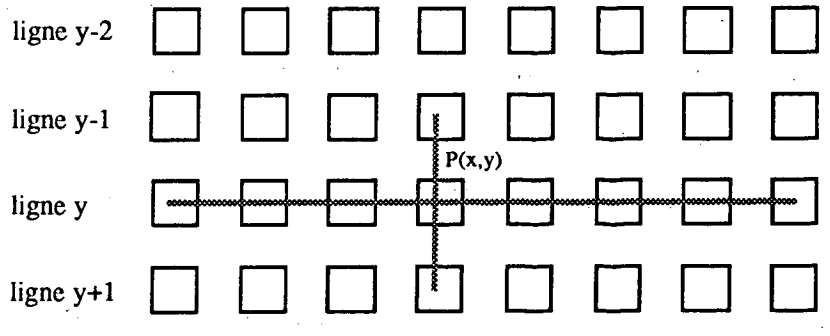

fig.16 Topologie pour l'évaluation de Rseg2; chaque ligne de processeurs représente une ligne de l'image et chaque colonne un vaisseau.

Le réseau final pour l'évaluation du programme de segmentation est obtenu en superposant les deux topologies; on obtient donc ici le réseau de la figure 16.

6. Conclusion.

Nous avons décrit un problème de corrélation d'images stéréoscopiques dont nous avons spécifié puis dérivé la partie segmentation. Ce travail montre que des techniques formelles de construction de programmes sont à même de résoudre des problèmes difficiles. Cela a été possible grâce aux propriétés du formalisme GAMMA dans lequel les programmes ont été dérivés. Le non-déterminisme du modèle de calcul, l'absence de flot de contrôle explicite et de récursivité, ainsi que l'utilisation du multi-ensemble comme structure de données permet de limiter fortement le contrôle dans les programmes; cela facilite d'autant l'obtention du programme à partir des propriétés logiques du problème. De plus, la spécification en logique du 1er ordre a permis d'approfondir notre compréhension de la segmentation: par rapport aux travaux précédents [Camillerapp 87], les propriétés liant deux points d'un même segment ont été affinées. Il est à noter aussi que les programmes dérivés n'occupent que quelques dizaines de lignes. Ce résultat est à comparer aux programmes PL/1 précédemment obtenus: plusieurs centaines de lignes dont la correction n'est pas prouvée et dont la mise au point est d'autant plus difficile que le volume des données est important. Enfin, le non-déterminisme de GAMMA et la propriété de localité des calculs autorise une interprétation parallèle: nous avons décrit dans la partie 5.1 une mise en œuvre possible. Cependant, il va de soi que cette mise en œuvre doit être vue comme une expérimentation plutôt que comme une implémentation réaliste. Le langage ne peut être utilisable en pratique si des techniques d'analyse statique ne sont pas mises au point afin de réduire la combinatoire engendrée. Le paragraphe 5.2 contient des premières suggestions; nous portons actuellement nos efforts dans cette direction.

Il est intéressant de comparer GAMMA à l'approche de Dijkstra-Gries, pour la construction de programmes itératifs. Dans leur formalisme, une boucle a la forme suivante:

$$* [C(x) \rightarrow \Psi(x)]$$

La sémantique associée est la suivante: tant que C(x) est vraie, appliquer Ψ à x. On peut

l'exprimer ainsi dans le formalisme GAMMA:

$$\Gamma(R,A) : \{x_0\} \quad \textbf{avec}$$
$$R(y) = C(y)$$
$$A(y) = \{\Psi(y)\}$$

où x_0 est la valeur initiale de la variable x. Ce programme exprime le fait que si l'unique élément y du multi-ensemble argument est tel que C(y), alors il est remplacé par $\Psi(y)$.

Plus généralement, une boucle composée de n commandes gardées agissant sur k variables, dénotées par le vecteur $X=(x_1, \ldots, x_k)$ s'écrit:

$$* [\ C_1(X) \rightarrow \Psi_1(X),$$
$$\ldots$$
$$C_n(X) \rightarrow \Psi_n(X)\ ]$$

Elle se traduit en GAMMA par:

$$\Gamma(\ (C_1,\Psi_1), \ldots, (C_n,\Psi_n)\ )$$

Bien sûr, aucun parallélisme n'est possible car le multi-ensemble argument ne possède qu'un élément, qui représente l'état global du programme.

Dans [Chandy 88], Chandy et Misra ont proposé une théorie, appelée UNITY, basée sur un modèle de calcul et un système de preuve associé. Le but est de pouvoir développer des programmes qui peuvent être mis en œuvre sur différentes architectures, aussi bien parallèles que séquentielles. Le développement se déroule en deux phases: la première phase consiste à obtenir un programme correct, indépendamment de la mise en œuvre; la deuxième phase vise à translater le programme obtenu sur l'architecture cible. Un programme UNITY se compose d'un ensemble d'instructions qui peut prendre une forme énumérée: $< Aff_1 \ [] \ldots [] \ Aff_n >$ ou quantifiée: $< [] \ i : 1 \leq i < n :: Aff_i >$; Aff_i peut être une instruction d'affectation simple (x:=E), multiple ($x_1:=E_1 \ || \ldots || \ x_k:=E_k$ ou $x_1, \ldots, x_k := E_1, \ldots, E_k$) ou conditionelle ($x:=E_1$ if $b_1 \sim \ldots \sim E_k$ if b_k). L'exécution d'un tel programme équivaut à une séquence infinie d'exécutions d'instructions d'affectation: à chaque étape, une instruction est choisie de façon non-déterministe dans l'ensemble et elle est exécutée. L'exécution peut s'arrêter quand un point fixe est atteint, c'est-à-dire quand l'exécution de n'importe quelle instruction ne modifie

pas l'état du programme. Prenons l'exemple d'un programme de tri par échanges:

Program tri

assign < [] i : 1≤i<n :: A[i], A[i+1] := A[i+1], A[i] **if** A[i]>A[i+1] >

end

Ce programme se compose de n-1 instructions d'affectation (quantifiées sur i). On imagine aisément sur cet exemple une exécution parallèle des opérations d'échange.

L'approche prise par Chandy dans sa théorie est voisine de celle de GAMMA. Dans les deux cas, il y a deux phases dans le développement des programmes: d'une part l'obtention d'un programme correct, sans faire intervenir des problèmes de mise en œuvre; d'autre part, la translation de ce programme sur une machine cible. Plusieurs points communs rapprochent les deux formalismes: leur schéma d'exécution est non déterministe et on ne trouve pas de flot de contrôle explicite. Cependant, le parallélisme dans UNITY est moint immédiat qu'il ne l'est dans GAMMA. Lors de la mise en œuvre d'un programme UNITY sur une machine parallèle, deux solutions sont possibles:

- soit la machine est synchrone, et il faut alors rechercher explicitement le parallélisme potentiel du programme, c'est-à-dire regrouper dans une instruction d'affectation multiple les opérations qui peuvent s'exécuter en parallèle. En GAMMA, le parallélisme est immédiat et implicite: peuvent s'exécuter en parallèle des réactions agissant sur des n-uplets disjoints.

- soit la machine est asynchrone, et on peut alors répartir les instructions d'affectation sur les processeurs. Cependant, les opérations effectuées sur les données n'étant généralement pas locales, on est confrontés à des problèmes de partage de données, qui limitent sensiblement le parallélisme. Ce n'est pas le cas en GAMMA, parce qu'un programme est construit de façon à obtenir des opérations les plus locales possibles.

Ces problèmes sont dûs au fait que les structures de données et de contrôle dans UNITY restent impératives, tandis que GAMMA utilise les multi-ensembles et un modèle de calcul adapté à ce type de donnée.

Remerciements

Nous tenons à remercier vivement J.-P. Banâtre, directeur de recherche et chef du projet Langages et Systèmes Parallèles à l'IRISA, ainsi que D. Le Metayer, membre de ce projet, pour leurs précieux conseils dans l'élaboration de cet article. Nous remercions également J. Camillerapp dont les travaux nous ont permis de dériver l'algorithme présenté ici.

Références

[Banâtre 86] J-P. BANATRE, D. LE METAYER: *A new computational model and its discipline of programming*; Rapport de recherche INRIA n° 566, septembre 1986.

[Banâtre 87] J-P. BANATRE, A. COUTANT, D. LE METAYER: *A parallel machine for multiset transformation and its programming style*; Frontiers in computing, Amsterdam 9-11 décembre 1987, North-Holland.

[Banâtre 88] J-P. BANATRE, A. COUTANT, D. LE METAYER: *Parallel machines for multiset transformation and their programming style*; Informationstechnik, 2/1988, pp. 99-109.

[Camillerapp 87] J. CAMILLERAPP, I. LEPLUMEY, A. WALTER: *Acquisition d'un modèle tridimensionnel du réseau vasculaire cérébral à partir d'un couple stéréoscopique*; 6ème congrès AFCET Reconnaissance de formes et I.A., Antibes, 16-20 novembre 1987, pp. 341-349.

[Chandy 88] K.M. CHANDY, J. MISRA: *Parallel Program Design*; Addison-Wesley Publishing Company, 1988.

[Coutant 89] A. COUTANT: *Synthèse de programmes dans le formalisme GAMMA*; Thèse, Université de Rennes 1, à paraître (1989).

[Creveuil 87] C. CREVEUIL: *Mise en œuvre distribuée de GAMMA sur iPSC*; Rapport de DEA, Université de Rennes 1, 1987.

[Dijkstra 76] E.W. DIJKSTRA: *A discipline of programming*; Prentice-Hall, Englewood Cliffs, N.J., 1976.

[Faugeras 88] O. FAUGERAS: *Quelques pas vers la vision artificielle en 3D*; TSI, 7,6, 1988, pp. 547-590.

[Gries 81] D. GRIES: *The science of programming*; Springer Verlag, New York, 1981.

[Leplumey 86] I. LEPLUMEY: *Une approche structurelle de la segmentation d'images et de la squelettisation*; Thèse de Docteur-Ingénieur, Université de Rennes 1, 1986.

[Long 85] P. LONG, M. BERTHOD, G. GIRAUDON: *Passage de primitives de bas niveau à des primitives de haut niveau dans le problème de mise en correspondance stéréo*; 5ème congrès AFCET Reconnaissance de formes et I.A., Grenoble, 27-29 novembre 1985, pp. 1233-1240.

[Marr 82] D. MARR: *Vision : A computational Investigation into the Human Representation and Processing of Visual Information*; W.H. Freeman and Co, Oxford, England, 1982, pp. 111-159.

[Moguérou 88] G.MOGUEROU: *Dérivation d'une application image dans le formalisme GAMMA*; Rapport de DEA, Université de Rennes 1, 1988.

LISTE DES DERNIERES PUBLICATIONS INTERNES

PI 464 **VERS UN MODELE D'ECLAIREMENT REALISTE**
Pierre TELLIER, Kadi BOUATOUCH
66 Pages, Avril 1989.

PI 465 **COMPILATION OF FUNCTIONAL LANGUAGES BY PROGRAM TRANSFORMATION**
Pascal FRADET, Daniel LE METAYER
28 Pages, Avril 1989.

PI 466 **MODEL BASED DIAGNOSIS : A CASE STUDY IN VIBRATION MECHANICS**
Michèle BASSEVILLE, Albert BENVENISTE
26 Pages, Avril 1989.

PI 467 **DELAI DE COMMUNICATION ENTRE NOEUDS VOISINS SUR L'iPSC/2**
Patrice BURGEVIN, André COUVERT, René PEDRONO
16 Pages, Avril 1989.

PI 468 **OBSERVING GLOBAL STATES OF ASYNCHRONOUS DISTRIBUTED APPLICATIONS**
Jean-Michel HELARY
24 Pages, Avril 1989.

PI 469 **ON-LINE MODEL CHECKING FOR FINITE LINEAR TEMPORAL LOGIC SPECIFICATIONS**
Claude JARD, Thierry JERON
16 Pages, Mai 1989.

PI 470 **PROGRAMMING WITH MALI - THE INTERPRETATION OF PROLOG PROGRAMS**
Louis CHEVALLIER, Serge LE HUITOUZE, Olivier RIDOUX
16 Pages, Mai 1989.

PI 471 **VERS UNE PROBLEMATIQUE DE L'ALGORITHMIQUE REPARTIE**
JeanMichel HELARY, Michel RAYNAL
12 Pages, Mai 1989.

PI 472 **DERIVATION SYSTEMATIQUE D'UN ALGORITHME DE SEGMENTATION D'IMAGES - UN EXEMPLE D'APPLICATION DU FORMALISME GAMMA**
Christian CREVEUIL, Gersan MOGUEROU
46 Pages, Mai 1989.

Imprimé en France
par
l'Institut National de Recherche en Informatique et en Automatique